Thesis for the degree of Licentiate of Engineering

Nucleon-Nucleon Scattering in a Wave-Packet Formalism

Sean B.S. Miller

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2020

Nucleon-Nucleon Scattering in a Wave-Packet Formalism
Sean B.S. Miller

Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 (0)31-7721000

Cover: Phase shift for the partial-wave $^3P_0$-state in a thirty-two-dimensional wave-packet basis, see chapter 3. The image is produced using Python with data from a c++ code written by the author.

Chalmers Reproservice
Gothenburg, Sweden 2020

Nucleon-Nucleon Scattering in a Wave-Packet Formalism

Sean B.S. Miller
Department of Physics
Chalmers University of Technology

Abstract

In this thesis I analyse the prospect of leveraging statistical analyses of the strong nuclear interaction by using the wave-packet continuum discretisation (WPCD) method to efficiently compute nucleon-nucleon (NN) scattering observables on a graphics processing unit (GPU). The WPCD method gives approximate solutions to the S-matrix at multiple scattering energies at the cost of a single eigendecomposition of the NN channel Hamiltonian. In particular, I demonstrate and analyse the accuracy and inherent parallelism of the WPCD method by computing the most common NN scattering observables using a chiral Hamiltonian at next-to-next-to-leading order. I present an in-depth numerical study of the WPCD method and the GPU acceleration thereof. Additionally, I discuss which windows of opportunity are open for studying the strong nuclear interaction using data from few-nucleon scattering experiments.

Keywords: nuclear physics, two-nucleon scattering, chiral potentials

Contents

Chapter 1

Introduction

The driving principle in the physical sciences is to understand and predict *all* phenomena and properties of physical systems. In fundamental nuclear physics this principle manifests itself in the studies of atomic nuclei; how they are shaped, how they interact, their electric charge distribution, the quantum spin, binding energy, density, size, etc. These properties of atomic nuclei are governed by the electromagnetic, weak, and strong forces between the constituent nucleons, classified into protons and neutrons. Of these three fundamental forces, it is the strong force that binds nucleons into atomic nuclei. Theoretical studies of nuclei from first principles (*ab initio*) are based on two practices. Firstly, we numerically solve a quantum mechanical A-body problem defined by the Schrödinger equation. The methods used for such simulations are commonly referred to as ab initio many-body methods, a few examples of which are quantum Monte-Carlo [1], coupled-cluster [2], and the no-core shell model [3]. The numerical precision with which we can solve many-body equations has improved with increasing computing power to such an extent that we can now begin to systematically quantify and control the intrinsic method uncertainty, thus ushering in an era of precision nuclear physics [4]. Secondly, to set up a realistic A-body Hamiltonian for the Schrödinger equation we need a theoretical description of the strong nuclear force. Any such descriptive invention for the strong force will here be referred to as a *model*.

There exists many models for the strong nuclear force. Some are better than others, and they can vary wildly in their predictive capability and region of applicability. Historically, finding a nuclear interaction model began in 1935 with the phenomenological meson exchange theory by Yukawa[5]. Since then the theory of nuclear forces has gone through several stages, see e.g. [6] for historical account,

until arriving at the effective field theory (EFT) [7] approach widely used today. An EFT maintains the symmetries of an underlying quantum field theory (QFT) while introducing relevant degrees of freedom and an expansion parameter in which to expand observables at a chosen momentum scale, i.e. a kind of perturbation theory. Modern EFTs of the strong force strive to build upon the Standard Model of particle physics (SM), wherein the strong interaction between quarks and gluons, the constituents of nucleons, interact is described by quantum chromodynamics (QCD). This fundamental theory rests on the colour-SU(3) gauge symmetry of quarks and gluons to derive a description of their mutual interaction in a QFT framework [8, 9]. However, it is problematic to make predictions at the nuclear energy-scale directly using QCD since the low-energy interactions between quarks and gluons are infinitely complex and non-perturbative. Instead, chiral EFT (χEFT) introduces an effective scheme [10, 11] for describing inter-nucleon interactions via pion exchanges at the cost of introducing infinitely many unknown low-energy constants (LECs) [12–17]. Hence pions and nucleons are treated as new effective degrees of freedom, while the complicated dynamics of quarks and gluons at low energy are replaced by effective interaction vertices whose strengths are governed by the LECs, which must be determined a posteriori. The infinite number of terms in the χEFT expansion must be truncated. A power counting scheme offers to group χEFT terms in orders of decreasing importance; a prerequisite for meaningful truncation of any perturbation series. With every new order in the power series there are several new LECs, thus making the model increasingly complex. It is worth mentioning that there exist some fundamental problems with χEFT that have yet to be solved (see e.g. [4] for a current account), especially in regard to the details of the power counting [18–22].

From a practical viewpoint, the χEFT approach may be considered simply as a model with several unknown parameters. To determine model parameters is a central topic in statistical inference and is referred to as parameter estimation. In Bayesian statistics, the approach to parameter estimation is to determine the *posterior* probability density function (pdf) $P(\boldsymbol{\alpha}|D)$ of the parameter values $\boldsymbol{\alpha}$ given some calibration data D. This is achieved by using Bayes' theorem,

$$P(\boldsymbol{\alpha}|D) = \frac{P(D|\boldsymbol{\alpha})P(\boldsymbol{\alpha})}{P(D)}\,, \tag{1.1}$$

which includes the following components: The *prior* $P(\boldsymbol{\alpha})$ encodes our belief about the values for $\boldsymbol{\alpha}$ before looking at the data D. The *evidence* $P(D)$ is independent of $\boldsymbol{\alpha}$ and it is not relevant in a parameter estimation problem. The *likelihood* $P(D|\boldsymbol{\alpha})$ is the pdf for the data given the model parameters. To evaluate the likelihood, we

must confront our model prediction with the data. This requires calculation of the relevant physical observables, often through numerical simulation. In the *ab initio* nuclear approach, such simulations can be computationally demanding for most observables. The parameter estimation problem is further complicated by the large number of parameters in χEFT models of the strong force. In conclusion, the computational bottleneck in parameter estimation resides in repeated likelihood evaluations. There are three common strategies to reduce this cost. Firstly, advanced parameter sampling such as through Markov-Chain Monte-Carlo [23] can reduce the number of times we need to evaluate a likelihood and still provide an accurate pdf. Secondly, replacing the numerical simulation with a fast and accurate emulator, e.g. a Gaussian process, will reduce the cost of each individual likelihood evaluation [24]. Thirdly, devising a computationally efficient method for computing observables will also reduce the cost of individual likelihood evaluations. This last option is the main focus in this thesis, and necessarily involves the study of a numerical *method* for calculating observables. However, numerical methods add both numerical error, and algorithmic inaccuracy and imprecision, to the model prediction of an observable.

Whether a method is viable for predicting nuclear observables is entirely dependent upon the method error relative to the error introduced by the strong force model *uncertainty*. In an additive model for uncertainties, we can relate a physical model $f(\boldsymbol{\alpha})$ to a measurement result z via

$$z = f(\boldsymbol{\alpha}) + \epsilon_f + \epsilon + e\,, \tag{1.2}$$

where e is the measurement error, ϵ is the *model discrepancy* term, and ϵ_f is the *method uncertainty* term. Note that the $\boldsymbol{\alpha}$-dependence of ϵ and ϵ_f is suppressed. The method uncertainty ϵ_f should not be large in comparison to e or ϵ, but nor does it need to be overly small. The biggest factor in method optimisation is reducing method quality by enhancing method uncertainty. The balance of quality versus speed should be decided by the model uncertainty and experimental error.
The truncation of the χEFT expansion introduces a model uncertainty. The upshot is that the theory itself provides us with a handle on the analytical form of the truncation error and its momentum dependence. This means that χEFT in principle permits us to do uncertainty estimates at *each order* of the power expansion, allowing systematic studies of χEFT model accuracy[25–27]. Documenting model uncertainty is an extensive process in fundamental nuclear theory and falls well outside the scope of this thesis. However, assuming such an account exists, it is necessary to document the method uncertainty.

The focus in this thesis is to explore the method uncertainty and computational cost of the wave-packet continuum discretisation (WPCD) [28] method for predicting nucleon-nucleon (NN) scattering observables. The NN scattering process is interesting given its direct connection to the NN interactions of χEFT and the extensive experimental dataset available for NN scattering observables. The numerically costly part of predicting such observables lies in the solution of the two-body Lippmann-Schwinger (LS) equation. The standard approach to solve the LS equation is by matrix-inversion [29] at each experimental beamline energy.
The WPCD method is a bound-state method for solving quantum scattering problems. It allows for highly efficient, parallel computable solutions to the LS equation [30] and three-body Faddeev equations [31]. Specifically, it divides the free continuum of momentum states into discrete bins called wave packets. In doing so there is an inherent coarse graininess that affects NN observable precision. I have performed an extensive analysis of the precision and the computational benefits WPCD may provide. I have utilised the parallel capability of graphics processing units (GPUs) to make use of the inherent parallelism of the WPCD method.

Chapter 2

Nucleon-Nucleon Scattering Theory

Scattering theory is the mathematical framework describing the dynamics of interacting quantum particles. In essence, it relates the quantum mechanical interaction potential to observables like cross sections. There are different governing equations to solve depending on the number of interacting particles, which for two particles is the Lippmann-Schwinger (LS) equation [32].

In this chapter I will summarise the most important steps for obtaining solutions to the LS equation and calculating elastic scattering observables. This will include definitions of standard parametrisations used in nuclear scattering theory and a brief review of a well-known matrix-inversion method [29] for numerically solving the LS equation.

2.1 The Lippmann-Schwinger Equation

The LS equation is derived from the time-independent Schrödinger equation,

$$(\hat{h}_0 + \hat{v})|\psi\rangle = E|\psi\rangle \,, \tag{2.1}$$

where $\hat{h}_0$ is the free (kinetic) part of the two-particle relative Hamiltonian, $\hat{v}$ is the two-particle potential operator, and the two-particle eigenstates $|\psi^-\rangle$ and $|\psi^+\rangle$ have energy E and will be referred to as in- and outbound scattering states, respectively. Formally, the Schrödinger equation can be rewritten into the LS equation (see e.g.

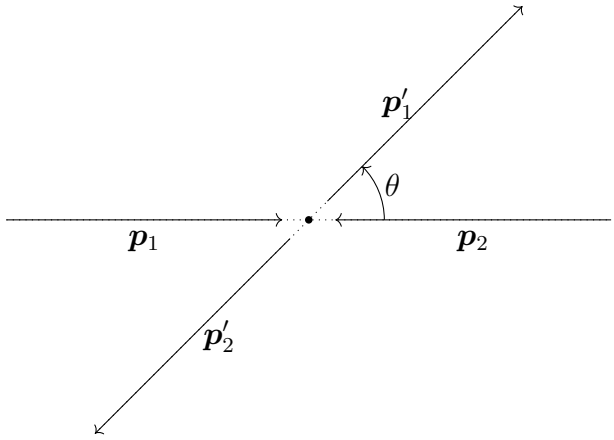

Figure 2.1: Initial ($\boldsymbol{p}_i$) and final ($\boldsymbol{p}_i'$) momenta of two colliding particles $i = 1, 2$ with scattering angle θ in the centre-of-mass coordinate system.

[33] for a detailed derivation)

$$|\psi^{\pm}\rangle = \frac{1}{E - \hat{h}_0 \pm i\epsilon}\hat{v}|\psi\rangle + |\phi\rangle \,, \tag{2.2}$$

where $|\phi\rangle$ is an eigenstate of $\hat{h}_0$ which I will refer to as a plane-wave state, and $\pm i\epsilon$ is an infinitesimal complex rotation introduced to deal with the singularity $\hat{h}_0(\hat{v}|\psi\rangle) = E(\hat{v}|\psi\rangle)$.

The LS equation can be expressed in terms of the transition matrix T, which relates an outbound scattering state $|\psi^+\rangle$ with a plane-wave state $|\phi'\rangle$ via the definition

$$T(\boldsymbol{q}', \boldsymbol{q}) \equiv \langle\phi'|\hat{v}|\psi^+\rangle \,, \tag{2.3}$$

where $|\phi'\rangle$ and $|\psi^+\rangle$ have momenta $\boldsymbol{q}'$ and $\boldsymbol{q}$, respectively, which are defined as the relative momenta of the two particles in the centre-of-mass system (c.m.s.) reference frame,

$$\boldsymbol{q} \equiv \frac{\boldsymbol{p}_1 - \boldsymbol{p}_2}{2} \quad , \quad \boldsymbol{q}' \equiv \frac{\boldsymbol{p}_1' - \boldsymbol{p}_2'}{2} \,, \tag{2.4}$$

with $\boldsymbol{p}_i$ and $\boldsymbol{p}_i'$ being the particles' momenta in the c.m.s. as shown in figure 2.1. In the following, I will suppress the superscript (+) and only consider outbound scattering states.

We can obtain a LS equation for the T-matrix by multiplying equation 2.2 by $\langle\phi'|\hat{v}$ from the left and inserting the definition in equation 2.3. These yields

$$T(\boldsymbol{q}', \boldsymbol{q}) = \langle\phi'|\hat{v}|\phi\rangle + \langle\phi'|\hat{v}\hat{g}_0(E)\hat{v}|\psi\rangle\,; \quad \hat{g}_0(E) \equiv \frac{1}{E - \hat{h}_0 \pm i\epsilon}\,, \tag{2.5}$$

where $\hat{g}_0$ is called the free resolvent.

I introduce a change in notation, $|\boldsymbol{q}\rangle = |\phi\rangle$ where $\boldsymbol{q}$ is as before, since this is a more common and convenient notation for plane-wave momentum-states. We can insert the identity operator $\mathbb{1} = \int \mathrm{d}^3p\,|\boldsymbol{p}\rangle\langle\boldsymbol{p}|$ to cast the LS equation in integral form,

$$T(\boldsymbol{q}', \boldsymbol{q}) = \langle\boldsymbol{q}'|\hat{v}|\boldsymbol{q}\rangle + \int \mathrm{d}^3p\,\frac{\langle\boldsymbol{q}'|\hat{v}|\boldsymbol{p}\rangle}{E - E_p \pm i\epsilon} T(\boldsymbol{p}, \boldsymbol{q})\,, \tag{2.6}$$

where $E_p \equiv \frac{p^2}{2\mu}$ is the kinetic energy of $|\boldsymbol{p}\rangle$. Equation 2.6 is an inhomogeneous Fredholm equation of the second kind, the solution of which is a studied in Fredholm theory.

An important aspect of the LS equation is that it can also be written using a resolvent for the full system Hamiltonian $\hat{h} \equiv \hat{h}_0 + \hat{v}$, i.e.

$$|\psi^{\pm}\rangle = \frac{1}{E - \hat{h} \pm i\epsilon}\hat{v}|\phi\rangle + |\phi\rangle\,. \tag{2.7}$$

Similar to the derivation of equation 2.5, we can use equation2.7 to express LS equation with the full resolvent $\hat{g}(E)$,

$$T(\boldsymbol{q}', \boldsymbol{q}) = \langle\phi'|\hat{v}|\phi\rangle + \langle\phi'|\hat{v}\hat{g}\hat{v}|\phi\rangle\,; \quad \hat{g} \equiv \frac{1}{E - \hat{h} \pm i\epsilon}\,. \tag{2.8}$$

The full resolvent $\hat{g}$ can only be calculated in an eigenbasis of the full Hamiltonian $\hat{h}$. These are unknown a priori, but I will show in chapter 3 that the wave-packet continuum discretisation method offers a way to approximate such scattering state projections of operators rather well.

2.1.1 Scattering observables and the T-matrix

Here, I will introduce the connection between scattering observables and the T-matrix in a top-down approach, starting with a general observable and then deriving an expression that can be linked to the T-matrix solution of the LS equation.

An observable will be spin-dependent if the scattering potential varies as a function of the particles' spin. Since nucleons are spin-$\frac{1}{2}$ fermions and since the nuclear force is spin-dependent, we need a formalism for observables that take spin-dependency into account. An observable expectation value of a general spin-dependent operator $\hat{O}$, which I hereafter refer to as a spin observable, can be expressed in terms of spin-state density matrices [33],

$$\langle \hat{O} \rangle = \frac{\mathrm{Tr}\{\hat{\rho}\hat{O}\}}{\mathrm{Tr}\{\hat{\rho}\}} \,, \tag{2.9}$$

where $\hat{\rho} \equiv \sum_{n=1}^{2} |\chi_n\rangle p_n \langle \chi_n|$ is the spin-state density operator with $\{|\chi_n\rangle\}_{n=1,2}$ as an arbitrary spin-state basis and p_n being the corresponding probability of occurrence. We let the spin-state basis be the usual helicity-state basis. In the asymptotic region, after scattering, the spin observable can be written in terms of the spin-scattering M-matrix,

$$\langle \hat{O} \rangle = \frac{\mathrm{Tr}\{M\hat{\rho}_i M^\dagger \hat{O}\}}{\mathrm{Tr}\{M\hat{\rho}_i M^\dagger\}} \,, \tag{2.10}$$

where ρ_i is the initial density matrix before scattering, which is the spin-state density matrix for pure helicity states.

It is common to parametrise M in terms of non-vanishing spin-momentum products after considering parity conservation, time-reversal symmetry, the Pauli principle, and isospin symmetry [34]. There exist several parametrisation conventions such as Hoshizaki [35], Wolfenstein [36–39], and Saclay [34] conventions. I use the Saclay parametrisation, where M is given by

$$\begin{aligned} M = \frac{1}{2}\Big[&(a+b) + (a-b)(\boldsymbol{\sigma}_1 \cdot \boldsymbol{n})(\boldsymbol{\sigma}_2 \cdot \boldsymbol{n}) + (c+d)(\boldsymbol{\sigma}_1 \cdot \boldsymbol{m})(\boldsymbol{\sigma}_2 \cdot \boldsymbol{m}) \\ &+ (c-d)(\boldsymbol{\sigma}_1 \cdot \boldsymbol{l})(\boldsymbol{\sigma}_2 \cdot \boldsymbol{l}) + e((\boldsymbol{\sigma}_1 + \boldsymbol{\sigma}_2) \cdot \boldsymbol{m})\Big] \,, \end{aligned} \tag{2.11}$$

where a, b, c, d, and e are the Saclay amplitudes, with momenta (see also figure 2.2)

$$\boldsymbol{l} \equiv \frac{\boldsymbol{q} + \boldsymbol{q}'}{|\boldsymbol{q} + \boldsymbol{q}'|} \,, \quad \boldsymbol{m} \equiv \frac{\boldsymbol{q} - \boldsymbol{q}'}{|\boldsymbol{q} - \boldsymbol{q}'|} \,, \quad \boldsymbol{n} \equiv \frac{\boldsymbol{q} \times \boldsymbol{q}'}{|\boldsymbol{q} \times \boldsymbol{q}'|} \,, \tag{2.12}$$

where $\boldsymbol{q}$ and $\boldsymbol{q}'$ are the relative momenta defined earlier, and $\boldsymbol{\sigma}_i$ are the Pauli spin matrices acting on nucleon $i = 1, 2$.

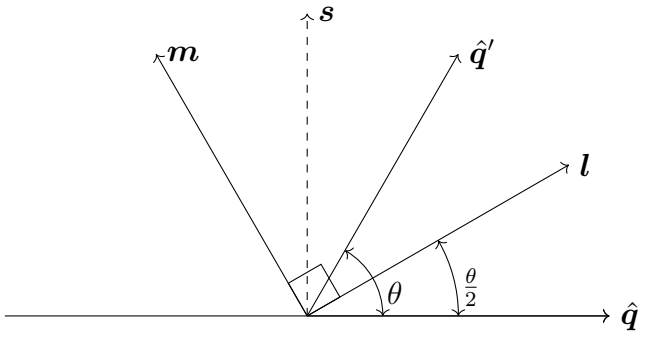

Figure 2.2: Non-relativistic particle-particle scattering kinematics in the c.m.s. [34] with scattering angle θ, where $\hat{\boldsymbol{q}}$ is the unit inbound relative momentum, $\hat{\boldsymbol{q}}'$ is the unit outbound relative momentum, and $\boldsymbol{s} \equiv \boldsymbol{n} \times \hat{\boldsymbol{q}}$. Vectors $\boldsymbol{l}$, $\boldsymbol{m}$, and $\boldsymbol{n}$ are defined in equations 2.12.

The total spin s is conserved in nucleon-nucleon interactions, motivating the use of two-nucleon spin states[1] $|s, m_s\rangle$. We therefore express the Saclay amplitudes in terms of spin-projections of the M-matrix,

$$M^s_{m_s,m_{s'}} \equiv \langle \boldsymbol{q}', s, m_{s'}|M|\boldsymbol{q}, s, m_s\rangle \,, \tag{2.13}$$

according to,

$$\begin{aligned}
a &= \frac{1}{2}(M^1_{11} + M^1_{0,0} + M^1_{1,-1}) \,, \\
b &= \frac{1}{2}(M^1_{11} + M^0_{0,0} + M^1_{1,-1}) \,, \\
c &= \frac{1}{2}(M^1_{11} - M^0_{0,0} + M^1_{1,-1}) \,, \\
d &= -\frac{1}{\sqrt{2}\sin(\theta)}(M^1_{1,0} + M^1_{0,1}) \,, \\
e &= \frac{i}{2\sqrt{2}}(M^1_{1,0} - M^1_{0,1}) \,, .
\end{aligned} \tag{2.14}$$

A partial-wave expansion (PWE) is at method used to express states in terms of angular momentum states. This is very useful since total angular momentum $J = s+l$ is conserved in nature, where l is the orbital angular momentum. Furthermore, the nuclear tensor-force couples angular momenta $l = J + 1$ and $l' = J - 1$, and thus

[1] It is not uncommon to use a helicity basis either.

the nuclear force can only couple 6 different combinations of partial-wave states for a given J. This truncates the PWE significantly. The $M^s_{m_s,m_{s'}}$-matrices are expressed as partial-wave expansions [33] (see also e.g. [40] for a pedagogical review),

$$\begin{aligned} M^s_{m_s,m_{s'}} = \frac{2\pi}{iq}\sum_{J,l,l'}[1-(-1)^{l+s+\tau}]Y^{l'}_{m_s-m_{s'}}(\theta,\phi)\sqrt{\frac{2l+1}{4\pi}} \\ \times \langle l', s, m_s - m_{s'}, m_{s'}|l', s', j, m_s\rangle \\ \times \langle l, s, m_s, 0|l, s, j, m_s\rangle \\ \times [\langle q', l', s, J|S|q, l, s, J\rangle - \delta_{l,l'}] \ , \end{aligned} \tag{2.15}$$

where the second and third rows are Clebsch-Gordan coefficients, J is the total angular momentum, l and l' is the orbital angular momenta of the inbound and outbound states respectively, τ is the azimuthal isospin projection, and $Y^l_m(\theta,\phi)$ is an azimuthal spherical harmonic. The angle θ denotes the scattering angle as before while ϕ is the rotation angle of $\boldsymbol{q'}$ around the inbound momentum $\boldsymbol{q}$, but cylindrical symmetry allows us to set $\phi = 0$. For the S-matrix we introduce a compact notation,

$$S^{sJ}_{ll'}(q', q) \equiv \langle q', l', s, J|S|q, l, s, J\rangle \ , \tag{2.16}$$

with which we define the S-matrix in terms of the T-matrix as [33]

$$S^{sJ}_{ll'}(q', q) = \delta^3(q' - q) - 2\pi i\delta(E' - E)T^{sJ}_{ll'}(q', q) \ , \tag{2.17}$$

where δ is the Kronecker delta. Here, the T-matrix does not depend on the direction (θ, ϕ) of $\boldsymbol{q'}$, so we can express the partial-wave LS equation for the T-matrix by integrating out all angular dependence in equation 2.6,

$$T^{sJ}_{ll'}(q', q) = \langle q'|\tilde{v}^{sJ}_{ll'}|q\rangle + \frac{2}{\pi}\int_0^\infty dq'' \ \langle q'|\tilde{v}^{sJ}_{ll'}\hat{g}_0|q''\rangle T(q'', q') \ , \tag{2.18}$$

where we have also introduced the corresponding partial-wave potential operator $\tilde{v}^{sJ}_{ll'}$. The potential expectation values can be found in works like e.g. [41].

The cross sections used to study the wave-packet formalism in this thesis will be the total and differential neutron-proton cross sections, σ_{tot} and $\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}$ respectively. These are given in terms of the Saclay amplitudes by [34],

$$\sigma_{\text{tot}} = \frac{2\pi}{q}\,\text{Im}\left[a(\theta = 0) + b(\theta = 0)\right] \ , \tag{2.19}$$

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} = \frac{1}{2}\left(|a|^2 + |b|^2 + |c|^2 + |d|^2 + |e|^2\right) \ . \tag{2.20}$$

2.1.2 Phase-shift parametrisation of the S-matrix

It is common in scattering theory to use a phase-shift parametrisation of the S-matrix. In elastic scattering, the S-matrix is an $n \times n$ unitary matrix that can be fully parametrised in terms of $2n-1$ parameters. The parameters are called phase shifts, written here as δ^{sJ}. For an uncoupled transition, i.e. $l = l'$, the parametrisation is given by,

$$S^{sJ}_{\text{uncoupled}} = e^{2i\delta^{sJ}} \, . \tag{2.21}$$

Due to the nuclear tensor force we must account for angular momentum coupling, i.e. $\langle q'|\tilde{v}^{1J}_{ll'}|q\rangle \neq 0$ for $l \neq l'$. There are two common S-matrix parametrisations in this case, referred to as the Blatt-Biedenharn [42] and the Stapp [43] parametrisations. The Blatt-Biedenharn parametrisation utilises the fact that the elastic S-matrix is unitary and therefore diagonalisable by a unitary transformation U,

$$S^J_T = U^{-1} \begin{pmatrix} e^{2i\delta^{1,J}_-} & 0 \\ 0 & e^{2i\delta^{1,J}_+} \end{pmatrix} U \, , \tag{2.22}$$

where

$$U = \begin{pmatrix} \cos(\epsilon_J) & \sin(\epsilon_J) \\ -\sin(\epsilon_J) & \cos(\epsilon_J) \end{pmatrix} \, , \tag{2.23}$$

and we introduce the notation $\delta^{s,J}_{-/+}$ is the lower/upper phase-shift, and ϵ_J is the mixing angle. It is more common to use the Stapp phase-shift parametrisation, where the coupled S-matrix is expressed in terms of phase shifts and mixing angles as

$$S^J_T = U^{-1} \begin{pmatrix} \cos(2\bar{\epsilon}_J) & i\sin(2\bar{\epsilon}_J) \\ i\sin(2\bar{\epsilon}_J) & \cos(2\bar{\epsilon}_J) \end{pmatrix} U \, , \tag{2.24}$$

where

$$U = \begin{pmatrix} e^{2i\bar{\delta}^{1,J}_-} & 0 \\ 0 & e^{2i\bar{\delta}^{1,J}_+} \end{pmatrix} \, , \tag{2.25}$$

where the "bar" notation is often used to distinguish the Stapp and Blatt-Biedenharn conventions. The contrast between the two is that the Blatt-Biedenharn convention treats the mixing of states as occurring in an "outer" region, while the Stapp convention treats it in an "inner" region. The two conventions are related via the following

three equations,

$$\begin{aligned} \bar{\delta}_-^{1,J} + \bar{\delta}_+^{1,J} &= \delta_-^{1,J} + \delta_+^{1,J} \,, \\ \sin\Big(\bar{\delta}_-^{1,J} - \bar{\delta}_+^{1,J}\Big) &= \frac{\tan(2\bar{\epsilon}_J)}{\tan(2\epsilon_J)} \,, \\ \sin\Big(\delta_-^{1,J} - \delta_+^{1,J}\Big) &= \frac{\sin(2\bar{\epsilon}_J)}{\sin(2\epsilon_J)} \,. \end{aligned} \tag{2.26}$$

All phase shifts presented here will be in the Stapp (bar) convention, although the bar-notation will be neglected for the sake of simplicity.

2.2 Numerical Approach; Gaussian Quadrature

The LS equation must be solved numerically, with the exception of a few simple potentials, such as e.g. the finite well and delta-shell potentials. In a numerical approach it is common to use a matrix-inversion method, see e.g. [29], to obtain the T-matrix. The method involves numerically expressing the integral using Gaussian quadrature and then solving a linear system.

Using a Gauss-Legendre grid $\{k_i\}_{i=1}^n$ with momenta k_i and corresponding weights w_i, we can approximate the integral in equation 2.18,

$$T_{ll'}^{sJ}(q,q) = \langle q|\tilde{v}_{ll'}^{sJ}|q\rangle + \frac{2}{\pi}\sum_{i=1}^{n} w_i \langle q|\tilde{v}_{ll'}^{sJ}|k_i\rangle\langle k_i|\hat{g}_0|k_i\rangle T_{ll'}^{sJ}(k_i,q) \,, \tag{2.27}$$

where q is the on-shell momentum. We let $q_i \in \{k_1, k_2, \ldots, k_n, q\}$ such that the operators can be written in matrix form with row i and column j corresponding to q_i and q_j respectively, e.g.

$$(V_{ll'}^{sJ})_{ij} \equiv \langle q_i|\tilde{v}_{ll'}^{sJ}|q_j\rangle \,. \tag{2.28}$$

In this notation the on-shell T-matrix element is given by $(T_{ll'}^{sJ})_{n+1,n+1}$. We introduce a vector D with elements defined as

$$D_i \equiv \begin{cases} \frac{2w_i q_i^2 M}{\pi(k_i^2 - q^2)} & \text{if} \quad i \leq n \,, \\ -\sum_{i=1}^{n} \frac{2w_i q^2 M}{\pi(k_i^2 - q^2)} + \frac{i\pi q M}{2} & \text{otherwise} \,, \end{cases} \tag{2.29}$$

which contains the weights and the resolvent. We rewrite equation 2.27 in matrix form to get

$$F_{ll'}^{sJ} T_{ll'}^{sJ} = V_{ll'}^{sJ} \,, \tag{2.30}$$

where we have introduced the wave matrix $F^{sJ}_{ll'}$ whose elements are given by

$$(F^{sJ}_{ll'})_{ij} \equiv \delta_{ij} - (V^{sJ}_{ll'})_{ij} D_j \, . \tag{2.31}$$

Equation 2.30 can be solved numerically in two different ways. The first is by matrix inversion of $F^{sJ}_{ll'}$, which is usually discouraged in scientific computing due to the instability of matrix inversion algorithms [44]. The more advisable practice is to use algorithms for LU-decomposition and solve equation 2.30 as a set of linear equations.

2.2.1 Complexity

I start here by emphasising that I only focus on the evaluation of the T-matrix and no other parts of evaluating scattering observables, like e.g. evaluating each term of the PWE. This is because I am ultimately only interested in a method-to-method comparison. Therefore, I will only consider the evaluation of a single partial-wave T-matrix in the following.

To evaluate the numerical efficiency of an implementation of the matrix-inversion method above, we need to know the minimum number of arithmetic operations to obtain an on-shell T-matrix element $T^{sJ}_{ll'}(q,q)$. Assuming there are n_E on-shell scattering energies $\{E_i\}_{i=1}^{n_E}$ for which we would like to solve the LS equation, and we are using a Gauss-Legendre grid with n points in our quadrature approach, then at each energy E_i we will have to set up the wave matrix $F^{sJ}_{ll'}$ and then solve for the derived T-matrix element.

The wave matrix 2.31 construction complexity is dominated by the matrix-matrix product of the potential V with the resolvent g_0. Since the resolvent is diagonal, it is more efficient to multiply each row of V with the diagonal element of g_0, yielding $n+1$ scalar-vector multiplications that amount to a complexity of

$$\mathcal{O}(F) = 2(n+1)^2 + 2(n+1) \, , \tag{2.32}$$

where the factor of 2 is due to g_0 being complex.

The on-shell T-matrix element is obtained from the last element of the T-matrix, i.e.

$$\langle q|T|q\rangle = T_{n+1,n+1} = \sum_{i=0}^{n+1} F^{-1}_{n+1,i} V_{i,n+1} \, , \tag{2.33}$$

where we have expressed it using matrix inversion. Matrix inversion typically requires $\mathcal{O}(n^3)$ operations for an $n \times n$ matrix. A stable and highly efficient matrix-inversion routine is Cholesky decomposition which reduces the complexity to $\mathcal{O}\left(\frac{n^3}{3}\right)$, but is only applicable to Hermitian, positive-definite matrices. Except for routines such as Cholesky decomposition, we have mentioned that solving the linear system,

$$FT = V \,, \tag{2.34}$$

is more advisable. This can be done very efficiently in two steps. First, we perform a lower-upper (LU) decomposition, where a square matrix A is expressed as the product of a lower-triangular matrix L with an upper-triangular matrix U,

$$A = LU \,, \tag{2.35}$$

which requires $\mathcal{O}\left(\frac{2}{3}n^3\right)$ operations. The decomposition allows for $AX = B$ to be solved for each column of X using $\mathcal{O}(2n^2)$ operations. For complex matrices these numbers increase by a factor 4 for both routines.

The two steps above will give a total computational cost of

$$\mathcal{O}(T) = n_E \left(\frac{8}{3}(n+1)^3 + 10(n+1)^2 + 2(n+1)\right) \,. \tag{2.36}$$

This formula will serve as an efficiency benchmark towards the end of this thesis, in chapter 5, particularly when analysing the efficiency of the WPCD method.

Chapter 3

Wave-Packet Continuum Discretisation

Wave-packet continuum discretisation (WPCD) [28] is, as the name implies, based on the use of wave packets. These are wave packets in momentum space that form an underlying discrete basis of quantum momentum-states. In this chapter I will present the definition of wave packets in terms of *pseudostates* and *eigendifferentials*, and then combine the two concepts into *pseudostate eigendifferentials* and from there on refer to these as wave packets. Furthermore, I will review the practical aspects of using this method to solve the LS equation for elastic neutron-proton scattering.

3.1 Scattering states and pseudostates

I start with a note on the nomenclature used here. States of a countable basis, infinite or finite, that spans the free energy region of the Hamiltonian are referred to as *discrete* states, whereas states of an infinite and uncountable basis of same region are referred to as *continuous* states.

The purpose of NN scattering theory is to calculate scattering observables, which are expectation values of free inbound and outbound plane-wave states[1]. However, the WPCD method employs discrete eigenstates of the system Hamiltonian to represent scattering observables. Therefore, it is necessary to derive a theory for the representation of scattering observables of continuous states in a discrete basis. This

[1]Plane-wave states are continuous states with a Dirac normalisation, and can therefore not be represented in a finite basis.

will be the topic of focus here.

We start by defining a finite discrete eigenbasis Ψ of the system Hamiltonian $\hat{h} \equiv \hat{h}_0 + \hat{v}$, of dimension m, as $\Psi \equiv \{|\tilde{\psi}_n\rangle\}_{n=1}^{m}$, where the tilde-notation is used to differentiate between continuous and discrete states for now. The states of the basis are referred to as "pseudostates" of the Hamiltonian $\hat{h}$ [28] since the states do not have a Dirac normalisation. A pseudostate $|\tilde{\psi}_n\rangle$ with energy E_n is given normally by the Schrödinger equation as

$$\hat{h}|\tilde{\psi}_n\rangle = E_n|\tilde{\psi}_n\rangle \,. \tag{3.1}$$

This equation appears identical to a bound-state system. In sharp contrast, however, the energies E_n can be both negative and positive, depending on $|\tilde{\psi}_n\rangle$ being a bound or free pseudostate. Similar to a bound-state problem, we find the pseudostate by a basis transformation to a finite eigenbasis $\Phi \equiv \text{span}\{|\tilde{\phi}_n\rangle\}_{n=1}^{m}$ of a Hamiltonian we can diagonalise analytically. For scattering systems, we use the free Hamiltonian $\hat{h}_0$,

$$\hat{h}_0|\tilde{\phi}_n\rangle = E'_n|\tilde{\phi}_n\rangle \,, \tag{3.2}$$

such that the transformation coefficients $\langle\tilde{\phi}_i|\tilde{\psi}_n\rangle$ determine Ψ to fulfil

$$\sum_{i=0}^{N}(\hat{h} - E_n)|\tilde{\phi}_i\rangle\langle\tilde{\phi}_i|\tilde{\psi}_n\rangle = 0 \,. \tag{3.3}$$

In scattering theory, we are only interested in observables, which are expectation values. Given some state $|\Psi\rangle$, the expectation value of an operator $\hat{O}(\hat{h})$ that depends on $\hat{h}$ can be represented in a continuous eigenstate basis of the Hamiltonian $\hat{h}$ as

$$\langle\Psi|\hat{O}(\hat{h})|\Psi\rangle = \sum_{i=1}^{n_b} f(\epsilon_i)|\langle\Psi|\psi_i^b\rangle|^2 + \int_0^\infty \mathrm{d}E f(E)|\langle\Psi|\psi(E)\rangle|^2 \,, \tag{3.4}$$

where $|\psi_i^b\rangle$ are bound eigenstates of $\hat{h}$ with energy ϵ_i, $|\psi(E)\rangle$ are (free) continuous eigenstates of $\hat{h}$ with scattering energy E, and we have introduced[2] $f(e) \equiv \langle\psi(e)|\hat{O}(e)|\psi(e)\rangle$. Since equation 3.4 is evaluated only in terms of the quadratic forms $|\langle\Psi|\psi(e)\rangle|^2$, we do not have to deal with the issue of representing continuous eigenstates in a discrete basis. It is therefore possible to approximate the expectation value using pseudostates,

$$\langle\Psi|\hat{O}(\hat{h})|\Psi\rangle \approx \sum_{i=1}^{n_b} f(\epsilon_i)|\langle\Psi|\tilde{\psi}_i^b\rangle|^2 + \sum_{i=1}^{n} f(E_i)|\langle\Psi|\tilde{\psi}_i(E_i)\rangle|^2 \,, \tag{3.5}$$

[2]Depending on the sign of e, $|\psi(e)\rangle$ can be either a free or bound state here.

where we also introduce quadrature weights w_i such that

$$|\langle\Psi|\tilde{\psi}_i\rangle|^2 = w_i|\langle\Psi|\psi_i(E_i)\rangle|^2 \,. \tag{3.6}$$

Finding the weights w_i can be done using an equivalent quadrature (EQ) technique [45–49] that allows for the calculation of non-negative spectral densities. It can be shown that the weights do not depend on the state $|\Psi\rangle$ and are instead a sort of transformation coefficient between pseudostates and continuous eigenstates. Using equation 3.6 we can introduce the following approximate relation,

$$\langle\psi_i|\hat{O}|\psi_i\rangle \approx \frac{\langle\tilde{\psi}_i|\hat{O}|\tilde{\psi}_i\rangle}{\sqrt{w_i}} \,. \tag{3.7}$$

This equation will be important in the following since it approximates continuous expectation values in a finite basis. The problem now is calculating the weights w_i, which is a highly non-trivial problem [28, 46, 47] when using continuous states. This is where the use of eigendifferentials is extremely advantageous.

3.2 Pseudostate eigendifferentials

Historically, eigendifferentials were used [50–52] as an alternative to have normalised free states before the Dirac-delta was introduced to normalise continuous states. We define an eigendifferential as the energy integral of free continuous eigenstates $|\psi(E)\rangle$ of a Hamiltonian $\hat{h}$, over some energy range ΔE,

$$|\psi(E,\Delta E)\rangle \equiv \int_E^{E+\Delta E} \mathrm{d}E\, |\psi(E)\rangle \,. \tag{3.8}$$

The energy range $\mathcal{D} \equiv [E, E+\Delta E]$ will here be referred to as an energy *bin*. An eigendifferential basis is defined by having non-overlapping bins. A few properties and identities of eigendifferentials are:

- Eigendifferentials are orthogonal,

$$\langle\psi(E,\Delta E)|\psi(E',\Delta E')\rangle = \begin{cases} \Delta E & \text{if} \quad \mathcal{D} = \mathcal{D}', \\ 0 & \text{otherwise} \,. \end{cases} \tag{3.9}$$

  and normalisable.

- The Hamiltonian is diagonal in an eigendifferential basis,

$$\frac{\langle\psi(E',\Delta E')|\hat{h}|\psi(E,\Delta E)\rangle}{\Delta E} = \begin{cases} E+\frac{1}{2}\Delta E & \text{if} \quad \mathcal{D}=\mathcal{D}', \\ 0 & \text{otherwise}\,, \end{cases} \tag{3.10}$$

- Given a Hamiltonian with n_b bound eigenstates $|\psi_i^b\rangle$, any state $|\Phi\rangle$ can be expressed in a basis of eigendifferentials as,

$$|\Phi\rangle = \sum_{i=0}^{N_b} c_i^b|\psi_i^b\rangle + \sum_{j=0}^{\infty} c_j|\psi(\mathcal{D}_j)\rangle\,, \tag{3.11}$$

  where c_i^b and c_j are the bound- and free-state expansion coefficients, respectively, and $\mathcal{D}_j \equiv [E_j, E_j+\Delta E_j]$ is eigendifferential j. In the limit $\{\Delta E_j \to 0\,\forall\, j \in \mathbb{N}\}$, the expansion above becomes a continuous state expansion.

- From definition 3.8 we get the following relation for an arbitrary operator $\hat{O}$,

$$\langle\Psi(E)|\hat{O}|\psi(E')\rangle \approx \frac{\langle\Psi(E)|\hat{O}|\psi(E,\Delta E)\rangle}{\sqrt{\Delta E}}\,, \tag{3.12}$$

  where $E' \in [E, E+\Delta E]$ and $|\Psi(E)\rangle$ is an arbitrary free state.

It is evident from the points above that eigendifferentials form an L_2-basis. The summation of free eigendifferentials in equation 3.11 is infinite and requires truncation for numerical application.

Representing observables in a finite eigendifferential basis is akin to the theory of pseudostates. Therefore, we refer to eigendifferentials of a *finite* eigendifferential basis as pseudostate eigendifferentials. To represent observables with pseudostate eigendifferentials it is necessary to determine the equivalent quadrature weights. It can be reasoned [53–56] that on behalf of equations 3.7 and 3.12 that the EQ weights are approximately given by

$$w_i \approx \Delta E_i\,. \tag{3.13}$$

In conclusion, we have introduced a finite basis of pseudostate eigendifferentials on which we can approximately project scattering observables using the weights of a quadrature-type technique.

3.3 Wave packets

In WPCD, wave packets are a generalisation of pseudostate eigendifferentials. Similarl to Rubtsova et al. [28], I define a free wave-packet $|x_i\rangle$ as a normalised, pseudostate eigendifferential using equation 3.8 with momentum $q \equiv \sqrt{2\mu E}$ rather than energy E,

$$|x_i\rangle \equiv \frac{1}{N_i} \int_{\mathcal{D}_i} q \, \mathrm{d}q \, f(q) |q\rangle \, , \tag{3.14}$$

where N_i is a normalisation constant, $f(q)$ is a weight function, and $|q\rangle$ is the radial part of the plane-wave state $|\phi\rangle$. I use the notation $\mathcal{D}_i$ to represent either the energy range as used for the eigendifferentials above or the corresponding momentum range $\mathcal{D}_i = \{q_i, q_i + \Delta q_i\}$, where $q_i = \sqrt{2\mu E_i}$ and $q_i + \Delta q_i = \sqrt{2\mu(E_i + \Delta E_i)}$. Note that a state $|q\rangle$ has the following standard normalisation,

$$\langle q|q'\rangle = \frac{\delta(q - q')}{qq'} \, . \tag{3.15}$$

The weight function $f(q)$ is introduced since we are free to let $|x_i\rangle$ be pseudostate eigendifferentials of either the momentum operator $\hat{p}$ or the energy operator $\hat{h}_0$,

$$\hat{h}_0 |x_i\rangle = \left(E_i + \frac{1}{2}\Delta E_i\right) |x_i\rangle \, , \tag{3.16}$$

$$\hat{p} |x_i\rangle = \left(q_i + \frac{1}{2}\Delta q_i\right) |x_i\rangle \, , \tag{3.17}$$

where we used equation 3.10. In principle the choice of $f(q)$ will have negligible impact on any calculation using the WPCD method [28]. If we let $|x_i\rangle$ be momentum eigendifferentials, then the width ΔE_i in equation 3.9 will be replaced by the corresponding momentum bin width Δq_i. Defining energy or momentum eigendifferentials corresponds to setting $f(q) = \sqrt{\frac{q}{\mu}}$ or $f(q) = 1$, respectively. These two types of wave packets are referred to as energy and momentum wave-packets, respectively. Bear in mind that using momentum wave-packets will change the weights of the equivalent quadrature in equation 3.13 to be $w_i \approx \Delta q_i$.

The free wave-packet basis with an upper energy limit E_n resolves the identity operator exactly, i.e.

$$\mathbb{1} = \sum_{i=1}^{n} |x_i\rangle\langle x_i| \, . \tag{3.18}$$

Using the identity operator and equations 3.14 and 3.15, we can relate continuous and discrete representations of a given some operator $\hat{O}$,

$$\langle q'|\hat{O}|q\rangle \approx \sum_{i,j=1}^{n} \langle q'|x_i\rangle\langle x_i|\hat{O}|x_j\rangle\langle x_j|q\rangle \,, \tag{3.19}$$

where I have calculated the basis transformation coefficients,

$$\begin{aligned} \langle q|x_i\rangle &= \frac{1}{\sqrt{N_i}} \int_{\mathcal{D}_i} p \,\mathrm{d}p \, f(p)\langle q|p\rangle \\ &= \frac{f(q)}{q\sqrt{N_i}} \quad \text{where} \quad q \in \mathcal{D}_i \,. \end{aligned} \tag{3.20}$$

We get the approximation

$$\langle q'|\hat{O}|q\rangle \approx \frac{f(q)f(q')}{\sqrt{N_i N_j}} \frac{1}{q'q} \langle x_i|\hat{O}|x_j\rangle \,, \tag{3.21}$$

where $q' \in \mathcal{D}_i$ and $q \in \mathcal{D}_j$. Equation 3.21 allows us to use the free wave-packet representation when solving the LS equation.
As a final comment, note that to accurately represent an operator it is important to set a sufficiently high energy or momentum boundary for the last wave packet in the basis, e.g. the momentum boundary $Q_n = q_n + \Delta q_n$ for $|x_n\rangle$ in a basis of dimension n. For example, a basis truncation at $Q_n = 100$ MeV will omit all higher momentum interactions and consequently fail to reproduce scattering phase-shifts and observables in an NN scattering simulation.

3.4 The Lippmann-Schwinger Equation in a Wave-Packet Basis

In this section I will show how the theory of wave packets can by used to derive the WPCD method. In chapter 2 I mentioned that the LS equation for the T-matrix can be straightforwardly solved using equation 2.8 if we have obtained an eigenbasis of the system Hamiltonian $\hat{h} = \hat{h}_0 + \hat{v}$. Following my discussion on pseudostates in section 3.1, we can approximate an eigenbasis of $\hat{h}$ in terms of a finite eigenbasis of $\hat{h}_0$ by solving equation 3.3, i.e. performing an eigendecomposition of $\hat{h}$.

Solving for the T-matrix using equation 2.8 requires us to find the transformation coefficients $\tilde{C}_{ij} \equiv \langle\tilde{\phi}_i|\tilde{\psi}_n\rangle$. In a wave-packet formalism, this corresponds to $C_{ij} \equiv$

$\langle x_i|z_j\rangle$, where $|x_i\rangle$ are free wave-packets and $|z_i\rangle$ are eigendifferentials of the system Hamiltonian with energies ϵ_i,

$$\hat{h}|z_i\rangle = \epsilon_i|z_i\rangle \,, \tag{3.22}$$

and are referred to as *pseudostate wave-packets* in accordance with [28]. The transformation matrix C with elements C_{ij} then fulfils,

$$|z_i\rangle = \sum_{j=1}^{n} C_{ij}|x_j\rangle \,, \tag{3.23}$$

which allows us to write the Hamiltonian matrix $H \equiv \langle x_i|\hat{h}|x_j\rangle$ as

$$H = CDC^T \,, \tag{3.24}$$

where D is a diagonal matrix of energies ϵ_i.

The eigenvalues ϵ_i are known from the eigendecomposition (see left of figure 3.1), but the bin boundaries for the pseudostate wave-packet bins remain undefined. I introduce the notation $[\mathcal{E}_i, \mathcal{E}_{i+1}]$ to define a pseudostate wave-packet bin, such that $\epsilon_i \in [\mathcal{E}_i, \mathcal{E}_{i+1}]$. Referring to equation 3.10, we see that eigendifferential eigenenergies are bin mid-points, i.e.

$$E + \frac{1}{2}\Delta E = \frac{1}{2}\left(E_i + E_{i+1}\right) \,, \tag{3.25}$$

where E_i and E_{i+1} are the bin boundaries of the eigendifferential pseudostate. Similarly, we need the eigenvalues ϵ_i to be the mid-points of the energy bins. In principle, this makes it possible to define the bin boundaries of pseudostate wave-packets using the energies ϵ_i. However, we do not know the exact lowermost and uppermost bin boundaries $\mathcal{E}_0$ and $\mathcal{E}_{n+1}$. Therefore, we create approximate bin boundaries for the pseudostate wave-packet as

$$\begin{aligned} \mathcal{E}_0 &\equiv 0 \,, \\ \mathcal{E}_i &\equiv \frac{1}{2}\left(\epsilon_i + \epsilon_{i+1}\right) \,, \\ \mathcal{E}_n &\equiv \epsilon_n + \frac{1}{2}\left(\epsilon_{n-1} + \epsilon_{n-2}\right) \,, \end{aligned} \tag{3.26}$$

such that they yield approximate eigenvalues $\bar{\epsilon}_i$,

$$\bar{\epsilon}_i \equiv \frac{1}{2}\left(\mathcal{E}_i + \mathcal{E}_{i+1}\right) \approx \epsilon_i \,. \tag{3.27}$$

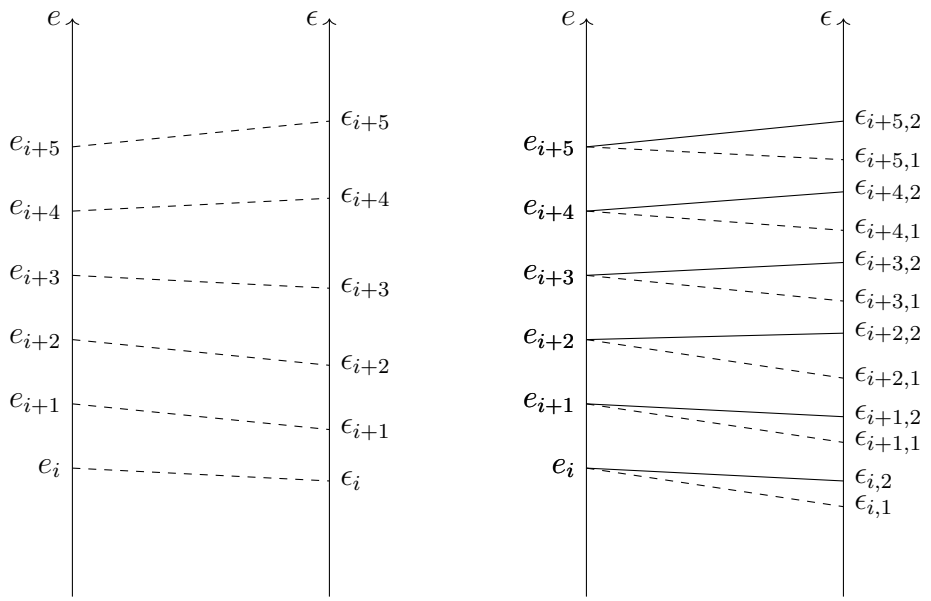

Figure 3.1: Left: Shift in free wave-packet energies $e_i \equiv E_i + \frac{1}{2}\Delta E_i$ to pseudostate wave-packet energies ϵ_i. Right: Splitting of degenerate free wave-packet energies e_i into energies $\epsilon_{i,1}$ and $\epsilon_{i,2}$ shown by solid and dashed lines, respectively. Note the a solid and a dashed line from two different energies e_i and e_j will not cross.

The quality of this approximation indirectly depends on the bin boundaries of the free wave-packet basis. Given a finite number of wave packets N_{WP}, narrower free wave-packet bins in a low-energy region improves the corresponding approximations above. However, it comes at the expense of poor approximation of high-energy physics. Increasing N_{WP} certainly resolves this but will also increase the computational cost of the Hamiltonian eigendecomposition. The effect of the wave-packet distribution and basis size on the method precision is a complex topic, the latter of which has been a major focus of the work presented here.

Here I mention an important side note on the construction of pseudostate wave-packet boundaries. In the case of coupled states due to the nuclear tensor force, the free wave-packet energies will split in the Hamiltonian eigendecomposition. This means a free wave-packet with energy $E_i + \frac{1}{2}\Delta E_i$ will give rise to two pseudostate wave-packet energies $\epsilon_{i,1}$ and $\epsilon_{i,2}$, corresponding to each coupled state labelled here as state 1 and 2. It is possible to show [28] that the split energies will not merge, i.e. $\epsilon_{i,2} < \epsilon_{j,1}\, \forall\, i < j$. Therefore, we use the boundary construction scheme in equation 3.26 such that the pseudostate wave-packets for coupled state l has boundaries

given by the energies $\epsilon_{i,l}$. Note it is very important to construct the Hamiltonian matrix with degenerate free wave-packet bases representing each coupled state, i.e. $|x_{1,i}\rangle = |x_{2,i}\rangle$, see right side of figure 3.1.

To solve the LS equation for the T-matrix using WPCD, I first define a T-matrix operator as the solution to the operator form of equation 2.8,

$$\hat{T}(E) = \hat{v} + \hat{v}\hat{g}(E)\hat{v} \,. \tag{3.28}$$

In WPCD, an approximate eigenbasis of the system Hamiltonian permits us to use equation 3.28 for representing the T-matrix elements for an on-shell energy E by

$$\bar{T}_{ij}(E) = \langle x_i|\hat{v}|x_j\rangle + \sum_{k=1}^{n}\langle x_i|\hat{v}|z_k\rangle\langle z_k|\hat{g}(E)|z_k\rangle\langle z_k|\hat{v}|x_j\rangle \,, \tag{3.29}$$

where I have defined $\bar{T}_{ij}(E) \equiv \langle x_i|\hat{T}(E)|x_j\rangle$. The wave-packet representation of the T-matrix relates to the usual T-matrix via equation 3.21,

$$T(q', q) \approx \frac{f(q)f(q')}{\sqrt{N_i N_j}}\frac{1}{q'q}\bar{T}_{ij}(E) \,. \tag{3.30}$$

There are two things to comment on in equation 3.29 in regard to wave-packet representations of the operators. Firstly, the potential term $\hat{v}$ is represented in a free wave-packet basis as,

$$\langle x_i|\hat{v}|x_j\rangle = \frac{1}{\sqrt{N_i N_j}}\int_{\mathcal{D}_i}\int_{\mathcal{D}_j} pp' \,\mathrm{d}p \,\mathrm{d}p' \; f(p)f(p')\langle p|\hat{v}|p'\rangle \,. \tag{3.31}$$

Typically, this integral is not analytically solvable, at least not for realistic nuclear potentials. There exist several approaches for solving it numerically, such as the Gaussian-quadrature method used in the matrix inversion method. However, since nuclear potentials vary mildly across a typical momentum-bin, it is sufficient to use a midpoint-approximation for evaluating the bin-integrals, and this turns out to have negligible effect in solving the LS equation. The mid-point approximation is given by

$$\langle x_i|\hat{v}|x_j\rangle \approx \frac{d_i d_j f(\bar{q}_i)f(\bar{q}_j)}{\sqrt{N_i N_j}}\langle \bar{q}_i|\hat{v}|\bar{q}_j\rangle \,, \tag{3.32}$$

where $d_i \equiv q_{i+1} - q_i$ is the momentum-bin width of $\mathcal{D}_i$, and $\bar{q}_i = \frac{q_i + q_{i+1}}{2}$ is the momentum bin mid-point. Clearly, this approximation drastically reduces computation

time for the potential matrices.

Secondly, from the definition of the full resolvent $g_i(E)$ in equation 2.8, it will singularities when the scattering energy $E \in \mathcal{D}_i$ is equal to any of the bin boundaries $\mathcal{E}_i$ and $\mathcal{E}_{i+1}$ for wave packet $|x_i\rangle$. To circumvent the singularities, we can use *energy averaging* by integrating equation 3.29 within the boundaries of $|x_i\rangle$ such that we use an energy averaged resolvent,

$$g_i^k \equiv \frac{1}{D_k} \int_{\mathcal{D}_k} \langle z_i | g(E) | z_i \rangle \, \mathrm{d}E = \frac{1}{D_k} \int_{\mathcal{D}_k} g_i(E) \, \mathrm{d}E \,, \tag{3.33}$$

where $D_k = E_{i+1} - E_i$ is the free wave-packet energy bin width. The result is,

$$g_i^k = \frac{\delta_{ij}}{D_i N_i} \left[W_{ki}^+ - W_{ki}^- \right] - \frac{i\pi}{D_k} \delta_{ik} \,, \tag{3.34}$$

where,

$$W_{ki}^{\pm} \equiv \sum_{k'=k}^{k+1} \sum_{i'=i}^{i+1} (-1)^{k-k'+i-i'} \left[q_{k'} \pm q_{i'} \right] \ln \left| q_{k'} \pm q_{i'} \right| \,. \tag{3.35}$$

The intermediate steps are shown in appendix A.

3.5 Numerical Implementation

An implementation of the WPCD method will require the following steps, in the order presented:

1. Choosing a distribution of the free wave-packet bin boundaries and weighting function $f(q)$ in equation 3.14. This is a non-trivial problem, and some choices are for example a uniform, Gauss-Legendre, or a Chebyshev distribution. Of these, the Chebyshev distribution is used in this thesis as it is a theoretically well-motivated choice [46, 47, 49] for approximating non-negative spectral densities based on the use of Chebsyhev's inequality. The Chebyshev distribution for $n+1$ points $\{y_k\}_{k=0}^n$ is defined by [28]

$$y_k = \alpha \tan\left(\frac{2k+1}{4(n+1)} \pi \right) \,, \quad k = 0, \ldots, n \,, \tag{3.36}$$

   where α is some scaling factor. In the case of wave packets, I let y_k be either the momentum or energy bin boundaries and e.g. $\alpha = 100$ MeV.

2. Evaluating the system Hamiltonian in the free wave-packet basis and performing an eigendecomposition (by diagonalising the Hamiltonian matrix) to get both eigenvalues and eigenvectors, according to equation 3.24.

3. Using the eigenvalues to construct new energy bins corresponding to pseudostate wave-packets, using equation 3.26.

4. Calculating the full resolvent in the pseudostate wave-packet basis using either equation3.33 or 3.34 . As mentioned, it is advantageous to use an energy-averaged resolvent to avoid resolvent singularities.

5. Solving the LS equation 3.29, which written in its matrix-representation is given by
$$\bar{T}_{ij}(E) = V_{ij} + (VC)_{ik} g_{kk}(E)(VC)_{jk} \,, \tag{3.37}$$
where $(VC)_{ik} \equiv V_{ij}C_{jk}$, $V_{ij} \equiv \langle x_i|\hat{V}|x_j\rangle$, $g_{kk} \equiv \langle z_k|g(E)|z_k\rangle$, and C is the transformation matrix obtained from the Hamiltonian eigendecomposition. Here I have used the non-energy-averaged resolvent.

6. Transforming the T-matrix from a discrete to a continuous basis using equation 3.21. In the case of energy averaging, this equation will only yield identical values for all momenta q within a bin $\mathcal{D}_i$.

3.6 Complexity and Parallelism

Similar to section 2.2.1 I below present the minimum number of floating-point operations (FLOP) a computer must perform in a single calculation of the LS equation using the WPCD method. I will use the same notation as before where n is the number of basis states (wave packets) and n_E are the number of on-shell energies of interest.

3.6.1 Sequential complexity

The calculation of observables only requires on-shell T-matrix elements, meaning only a single T-matrix element is required given an energy E (i.e. $\bar{T}_{ii}(E \in \mathcal{D}_i)$ in equation 3.29). If we omit the complexity of evaluating the potential matrix, the complexity of solving the LS equation is dominated by three parts; the Hamiltonian diagonalisation, two matrix-matrix products, and one case of addition, all of which I examine here:

1. For elastic NN scattering modelled with χEFT the system Hamiltonian will be real, so we need to diagonalise a square, symmetric matrix. This is efficiently done using QR factorisation or a divide-and-conquer algorithm, both of which are usually used together with Householder transformations. The divide-and-conquer algorithm has a complexity of $\mathcal{O}\left(\frac{8}{3}n^3\right)$ for getting both eigenvectors and values.

2. We have to calculate the matrix-matrix product of the potential matrix V and the coefficient matrix C. Square matrix-matrix multiplication involves $2n^3 - n^2$ FLOP in a sequential approach.

3. The last operation is the calculation of the LS equation. Given the diagonalisation of H and the matrix product VC we solve the equation for the on-shell wave-packet of interest, using equation 3.37. The sum in the equation,

$$\sum_{j=0}^{n}(VC)_{ij}g_{jj}(E \in \mathcal{D}_i)(VC)_{ji}\,, \tag{3.38}$$

   involves two multiplication operations per term in the sum, and the sum will give $n-1$ additions. There is one additional complex addition in calculating the T-matrix element. Furthermore, the operations here must be done for every on-shell energy, adding an overall factor of n_E. We end up with a complexity of $6n \times n_E$ FLOP. However, note that if we use energy averaging (g^i_{jj} rather than $g_{jj}(E \in \mathcal{D}_i)$) then we only require a single evaluation for each on-shell T-matrix element. Therefore $n_E = n$ is the largest value possible with energy averaging.

In conclusion, the complexity of solving the LS equation for an on-shell T-matrix element is

$$\text{WPCD} : \mathcal{O}(T) = 6n^3 - n^2 + 6n \times n_E\,, \tag{3.39a}$$

$$\text{WPCD}(n_E = n) : \mathcal{O}(T) = 6n^3 + 5n^2\,. \tag{3.39b}$$

3.6.2 Parallel complexity

I emphasise here, just as in section 2.2.1, that I will only focus on the parallel evaluation of a single partial-wave T-matrix and not other parallel strategies like e.g. evaluating each term of the PWE in parallel. In the following I will only consider the parallel evaluation of the T-matrix for a single partial wave.

The components of the FLOP model in equation 3.39a will all change when done in parallel. All three algorithms, the diagonalisation, matrix-matrix multiplication, and the resolvent-summation, can be made in parallel. Below I review the complexity of the WPCD method using parallel algorithms.

1. The parallel order Jacobi method is a parallel approach to the Jacobi eigenvalue algorithm; a method based on finding a similarity transformation of a matrix to its diagonal form by repeated Jacobi rotations (see e.g. [57]). For an $n \times n$ matrix, it can be shown that it has a complexity of $\mathcal{O}(n \log^2 n)$ to converge when done in parallel.

2. Parallelisation of matrix-matrix multiplication algorithms is a vast field of research. There exist very efficient methods like e.g. Cannon's algorithm [58], but these depend highly on hardware. However, a straightforward way parallelise this step is using a common divide-and-conquer approach, an example of which I will present in chapter 4. The divide-and-conquer approach has a complexity of $\mathcal{O}(\log^2(n))$ when performed in parallel.

3. The only part that changes from the sequential approach when adding the resolvent term is that we can do the multiplication per summation term in parallel, while the addition of the terms I choose to do in sequentially due to the relatively low complexity. Therefore, there are 2 multiplications per term which are done in parallel, and $n-1$ summation-terms done in serial, as well as adding the potential-term outside the summation. This results in a complexity of $\mathcal{O}(2(n+2))$. This summation can be done independently for each energy so no factor of n_E is included.

In conclusion the parallel WPCD complexity model is

$$\text{WPCD} : \mathcal{O}(T) = (n+1)\log^2(n) + 2(n+2)\,. \tag{3.40}$$

We could, likewise to WPCD, argue that the matrix inversion approach can also be done in parallel. However, parallel algorithms for solving linear systems are not so efficient (ref needed) for the small matrix sizes we tend to use (typically $n < 100$ wave packets).

In figure 3.2 I compare the complexity of the WPCD (serial and parallel), method with the matrix inversion method for a range of reasonable numbers of on-shell energies n_E. It is clear that at each basis size n, both algorithms for the WPCD method outperforms the matrix inversion method. As we see, even for a basis size $n \sim$

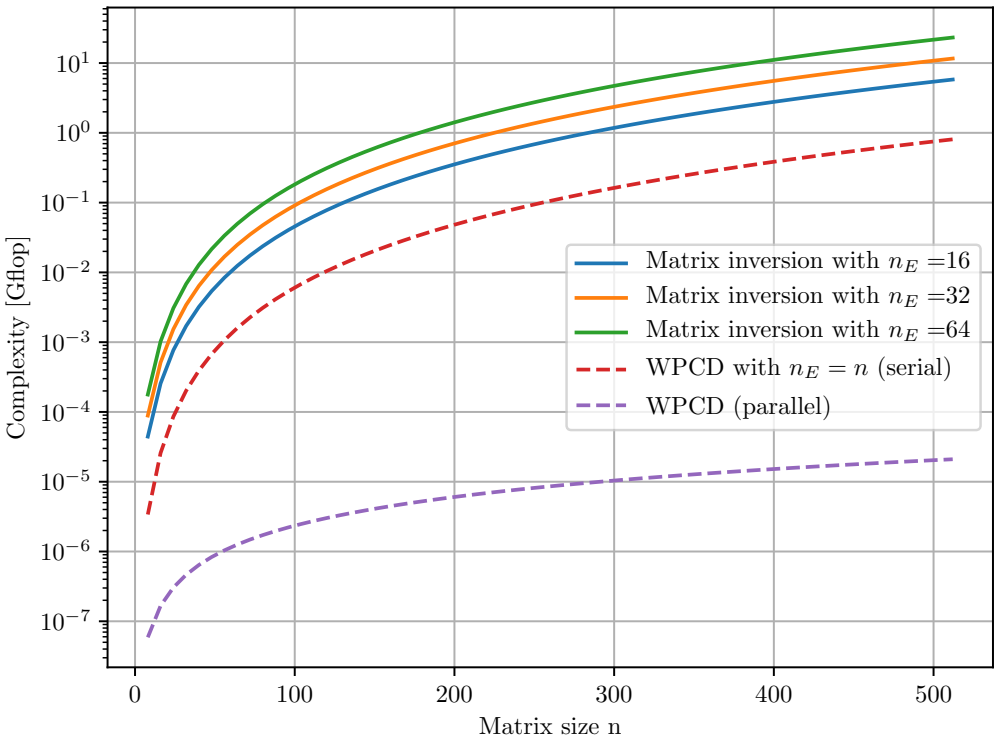

Figure 3.2: Complexity of the serial and parallel WPCD methods and matrix-inversion method versus the relevant matrix size n for each method, with n_E on-shell energy calculations. Here, n symbolises the basis size, in either number of Gauss-Legendre points or number of wave packets.

500, the parallel model should outperform the smallest matrix inversion calculation. This observation makes it interesting to analyse what we can achieve in the way of parallelisation of the WPCD method.

Chapter 4

Graphics Processing Units in Scientific Computing

The graphics processing unit (GPU) is a product from the race for faster computing hardware in the computer games industry. They are made to quickly generate images for computer displays, hence the term "graphics". Over the last decade the use of GPUs in scientific computing grew significantly, to such an extent that the industry has seen a profitable demand in the sciences. This creates a positive feedback loop; better GPU hardware and software increases their use in scientific computing, which increases demands on more and better GPU hardware and software. There is no apparent reason to avoid utilising GPUs for leveraging heavy numerical calculations, at least not until a new technology emerges or technological limitation halts GPU development.

In this chapter I will present the basics of scientific computing on a GPU. I explain the principle of a GPU and the differences from a CPU, and introduce an interface called CUDA for high-level programming languages with examples on how to use it efficiently with C++.

4.1 What is a GPU?

A GPU can be thought of as a CPU with multifold parallel processing. GPU architecture differs significantly from a CPU, but from a computing perspective there are foremost two key differences,

1. A GPU can compute hundreds of more tasks in parallel compared to a CPU.

2. A GPU operates at about half the processing frequency, called clock rate, of a CPU.

The design strategy behind a GPU is to prioritise the total *throughput* of calculations and not the rate of individual calculations, which is the case for a CPU. In table[1] 4.1 we see that while the GPU clock rates are significantly lower than the CPU clock rates, the number of cores, bandwidth, and floating-point operations per second (GFLOPS) is greater.

Table 4.1: GPUs and CPUs available at the C3SE [59] computing clusters. The GFLOPS are reported as two for one multiplication and addition for single-precision floating-point operations; divide the row by 2 for corresponding double-precision GFLOPS. See e.g. [60] for an extensive account. Note that a Cuda core and CPU core are not directly comparable.

GPU model	SMs	Cuda cores	Clock [MHz]	GFLOPS	Bandwidth [GB/s]
K40 [61]	15	2880	875	5040	288
T4 [62]	40	2560	1590	8100	320
V100 [63]	80	5120	1370	14028	900
CPU model	**Cores**	**FP32 units**	**Clock [MHz]**	**GFLOPS**	**Bandwidth [GB/s]**
Xeon E5-2650v3 [64, 65]	10	160	2600	832	68
Xeon Gold 6130 [66]	16	512	1900	1946	128

The two major producers of GPUs today are the Nvidia Corporation and Advanced Micro Devices, Inc. (AMD). Of the two, only Nvidia is currently developing significant support for general-purpose computing via their GPU interface named CUDA. CUDA allows the user to program the GPU using a few high-level languages like Python, C++, Fortran, etc. It also includes a large number of optimised routines for several common operations. There are several other similar interface platforms available, like DirectCompute and OpenCL, but these are written for arbitrary GPUs and are thus not optimised for specific hardware. CUDA is written specifically for Nvidia GPUs and will not work on other GPUs.

[1] An FP32 unit is a 32-bit processing unit capable of basic arithmetic operations, e.g. addition and subtraction, for 32-bit format numbers, also referred to as single-precision float-point format. A CPU core is composed of FP32 units.

4.1.1 GPU structure and CUDA

To write efficient GPU code it is important to know how a GPU operates; how it partitions, schedules, and executes code (see e.g. [67] for an extensive account). A piece of code written for parallel GPU calculation is called a kernel in CUDA. Each part of a parallel calculation is performed on a GPU thread. For example, if we wish to multiply n variables by some scalar a we can either do each multiplication sequentially, or we can write a kernel that starts n threads and telling thread i to multiply variable i by a. Use of concurrent operations is vital for maximal GPU performance.

GPU structure and code execution

An Nvidia GPU is divided into streaming processors (SPs)[2] called CUDA cores. One or several SPs are managed by a streaming multiprocessor (SM). An SM is a governing component (chip) that creates, manages, and schedules a block of threads. The SM divides the block into warps of threads executed on SPs. A warp is a group of 32 threads. The assignment of warps to SPs is managed by the SM. All the warps within an SM work on the same kernel, so they perform the same set of instructions. This is the SIMT (single-instruction multiple-threads) architecture of the GPU. All the threads within a warp execute simultaneously on an SP, but the warps/SPs within an SM execute independently. It is important to understand the GPU structure and to fully use the fact that an SP executes *all* 32 threads in a warp, regardless if some threads are not assigned instructions, to avoid wasting clock cycles.
A last note to be made here is that CUDA threads on SPs take significantly less time to initialise than a typical CPU thread.

Memory and memory transfers

The GPU central memory is called the global memory. Each thread has its own portion of the global memory called local memory, that is used when the thread runs out of registry space. Besides the global GPU memory, each chip (SM) has a 64kB memory cache. Because they are on-chip, these caches are $\sim$ 100x faster for threads to access. The cache can be divided and dedicated to warps/threads. All the threads within a block can then access the same section of the on-chip memory.

[2] AMD GPUs also have SPs that serve the same function but are different in design so the GPU structure presented here does not directly apply to AMD GPUs beyond SPs.

These cache segments are called shared memory. It is always best to minimise transfer between threads and the global memory by maximising the use of the shared memory. Therefore, it is wise make full use of the memory bandwidth to get everything needed for a kernel execution to the shared memory.

4.2 CUDA by example

It is difficult to learn any library or language without examples. I will therefore demonstrate CUDA programming in C++ for efficiently multiplying two matrices. It is apparent that to use GPUs efficiently we must both maximise bandwidth usage to the global memory and use the shared memory. Via examples I will first write a simple kernel using CUDA, followed by optimisation of global memory transfer, and end by improving the use of shared memory.

4.2.1 Writing a kernel for matrix-matrix multiplication

The GPU is a tool to alleviate the CPU workload, often called GPU acceleration. This typically means that a set of data on the CPU memory must be sent to the GPU for processing and returned to the CPU memory. In CUDA, the GPU and CPU are referred to as the *device* and *host*, respectively. Assume we have an array `A_h` on the host that we need on the device. In listing 4.1 we make an empty array `A_d` in the device global memory to fit `A_h`, and then transfer the content, i.e. the array elements.

Listing 4.1: Setup for host-to-device and device-to-host memory transfer.

```
#include <cuda_runtime.h>

int main(){
  double *A_h = new double [100];

  /* We fill array A_h with numbers on host */

  double A_d = NULL;
  cudaMalloc((void**)&A_d, sizeof(double)*100);
  cudaMemcpy(A_d, A_h, sizeof(double)*100, cudaMemcpyHostToDevice);

  /* Do something with A_d on device */

  cudaMemcpy(A_h, A_d, sizeof(double)*100, cudaMemcpyDeviceToHost);
```

```
  /* Do something with A_h on host */

  cudaFree(A_d);
  delete [] A_h;
  return 0;
}
```

I have here declared a host array `A_h` of 100 numbers in double precision, created a similar device array `A_d`, and transferred the content of `A_h` into `A_d`. The functions used here are part of the CUDA interface,

- `cudaMalloc`: Allocates device global memory space for an array of given length and numerical type.
- `cudaMemcpu`: Transfers content from one array to another, either both arrays on the host or device, or one array on either.
- `cudaFree`: Frees global memory occupied by an array on the device.

To operate on `A_d`, we will need to use a kernel. There are three different kinds of kernels denoted by the labels `host`, `device`, and `global`, which tell the compiler whether the code is to be called from the CPU, the GPU, or either, respectively. An example of a kernel can be made from a host function for multiplying the two matrices `A_h` and `B_h` on the host and storing the result in `C_h`, also on the host, presented in listing 4.2.

Listing 4.2: Simple row-major matrix-matrix multiplication on the host.

```
void multiply_matrices(double *A_h, double *B_h, double *C_h, int n)
   {
   for (int i=0; i<n; i++){
      for (int j=0; j<n; j++){
         for (int k=0; k<n; k++){
            C_h[i*n + j] += A_h[i*n + k]*B_h[k*n + j];
         }
      }
   }
}
```

The summation over index k is not thread-safe, so instead we parallelise the two outermost loops for indices i and j. In principle I let each thread have exclusive access to a matrix element [i,j] of `C_d`. Such a kernel is shown in listing 4.3.

Listing 4.3: Simple, parallel row-major matrix-matrix multiplication on the device.

```
__global__
void multiply_matrices_kernel(double *A_d, double *B_d, double *C_d,
    int n){

   int i = blockIdx.x * blockDim.x + threadIdx.x;
   int j = blockIdx.y * blockDim.y + threadIdx.y;

   for (int k=0; k<n; k++){
      C_d[i*n + j] += A_d[i*n + k]*B_d[k*n + j];
   }
}
```

The `global`-label means this is code that can be called directly from a CPU, but executed on the GPU as a kernel. The indices i and j are set by three CUDA functions,

1. `blockIdx`: The block index in which the current thread resides.

2. `blockDim`: The size of the block in number of threads.

3. `threadIdx`: The index of the thread within the block it resides.

There are two significant efficiency problems with this kernel. Firstly, each thread loops through the columns and rows of A_d and B_d, respectively. Regardless of whether we store matrices as row- or column-major, we will be reading at strided lengths through the global memory for elements of A_d or B_d. Looking up these elements cause asynchronous transfers to threads in a block, which wastes memory bandwidth. Secondly, we write to the global array C_d for every part of the loop summation. It is better to use a temporary summation variable in the thread registry. Then we write the registry variable to the global memory at the end.

4.2.2 Optimising memory usage

Optimising memory usage involves rewriting algorithms to allow for better hardware utilisation. We can reduce the number of times we read elements of `A_d` and `B_d` from the global memory by using sub-matrices that fit in the shared memory. In listing 4.4 I show how to transfer from global to shared memory, by letting each thread block copy a row of a larger matrix `A_d` in the global memory into the shared-memory array `A_row`,

Listing 4.4: Example of global-to-shared device memory transfer.

```
__global__
void copy_row_kernel(double *A_d, int row_length){
   int row_idx = blockIdx.x;
   int col_idx = threadIdx.x;

   __shared__ double A_row [row_length];

   A_row[col_idx] = A_d[row_idx*row_length + col_idx];

   __syncthreads();

   /* Do calculations using A_row */
}
```

Each block works on separate rows of `A_d`, and each block thread copies an element of the row. The `__shared__`-label tells the compiler that `A_row` is a pointer to an array of length `row_length` residing in the shared memory. We use the thread index to index the row-elements of `A_d`. It is important to synchronise the block threads when parallel-copying into the shared memory because otherwise a thread might read something in the shared memory before it has been written. Synchronising threads will cause a delay in the execution time of a block, but it is unavoidable if we need threads to access several elements of `A_row`.

The method of sub-matrices, referred to previously as the divide-and-conquer approach, is common in numerical linear algebra due to memory restrictions, and is not specific to GPU programming. For a matrix-matrix multiplication $C = A \times B$, where $A \in \mathbb{R}^{n\times l}$, $B \in \mathbb{R}^{l\times m}$, and $C \in \mathbb{R}^{n\times m}$, we divide the three matrices into smaller sub-matrices as shown in figure 4.1. We restrict the size such that three such sub-matrices can fit in the shared memory of a block. Each block is then given the same number of threads as there are elements in the sub-matrix, to allow for efficient global memory copying. To copy the sub-matrices, the only change we need to introduce in listing 4.4 is a delimiter such that blocks copy several, shorter rows of length $p \leq l$. I show this in listing 4.5, and for simplicity restrict ourselves to square parent- and sub-matrices. I have defined the width p of a block as `block_size`.

Listing 4.5: Example of global-to-shared device memory transfer with sub-matrices.

```
#define block_size 16

__global__
```

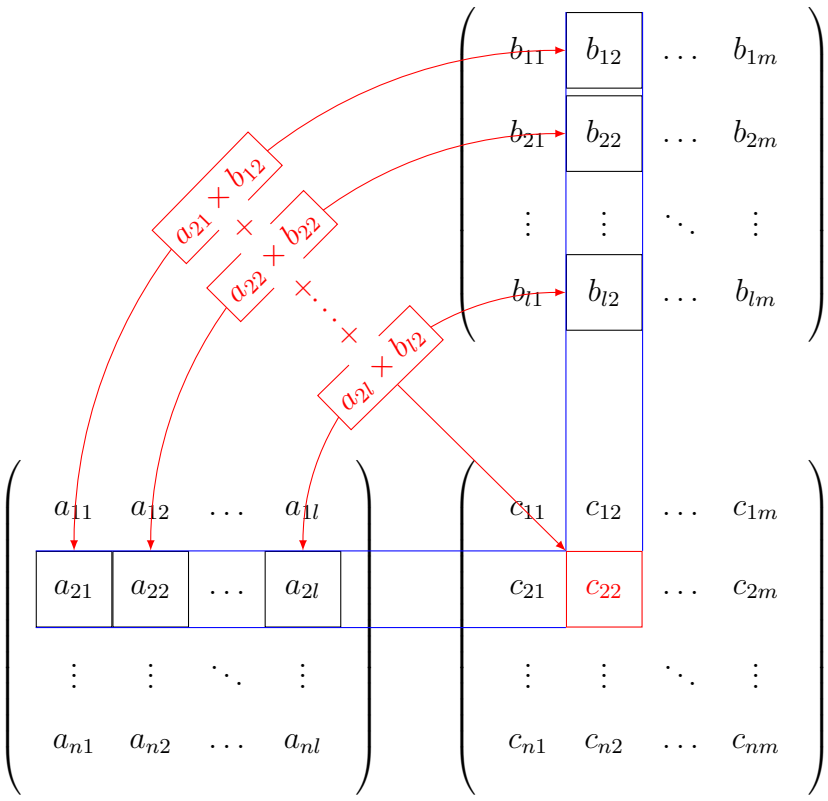

Figure 4.1: Matrix-matrix multiplication[68] with sub-matrices a_{ij}, b_{ij}, and c_{ij}.

```
void copy_block_kernel(double *A_d, int row_length){
   int row_block_idx = blockIdx.x*block_size;
   int col_block_idx = blockIdx.y*block_size;

   int row_idx = threadIdx.x;
   int col_idx = threadIdx.y;

   A_block_ptr = A_d[row_block_idx*row_length + col_block_idx]

   __shared__ double A_block [block_size][block_size];

   A_block[row_idx][col_idx] = A_block_ptr[row_idx*row_length +
      col_idx];

   __syncthreads();

   /* Do calculations using A_block */
```

```
}
```

In listing 4.6, matrix-matrix multiplication with sub-matrices is done by letting each thread perform a vector-vector multiplication between two sub-matrices of `A_d` and `B_d`. The resulting registry variable `C_sum` corresponds to an element of global memory array `C_d`. At the end we transfer back `C_sum` to `C_d`, using the `C_d` sub-matrix pointer `C_block_ptr`.

Listing 4.6: Shared-memory matrix-matrix multiplication using sub-matrices.

```
#define block_size 16

__global__
void multiply_matrices_kernel(double *A_d, double *B_d, double *C_d,
    int row_length){
   int row_block_idx = blockIdx.x*block_size;
   int col_block_idx = blockIdx.y*block_size;

   int row_idx = threadIdx.x;
   int col_idx = threadIdx.y;

   /* Pointer to the block's position in the global memory */
   C_block_ptr =  C_d[row_block_idx*row_length + col_block_idx];

   /* Variable kept on the thread registry */
   double C_sum = 0;

   for (int m=0; m<n/block_size; m++){
      /* Create A_block and B_block using copy_block_kernel,
       * remember to synchronise threads */

      for (int k=0; k<block_size; k++){
         C_sum += A_block[row_idx][k]*B_clock[k][col_idx];
      }

      __syncthreads();
   }

   /* Write/transfer C_sum back to global memory */
   C_block_ptr[row_idx*row_length + col_idx] = C_sum;
}
```

In figure 4.2 I show the time profile of the simple kernel in listing 4.3 and the shared-memory kernel in listing 4.6. The latter is faster by about one order of magnitude difference, showing that transfer time can be reduced significantly with

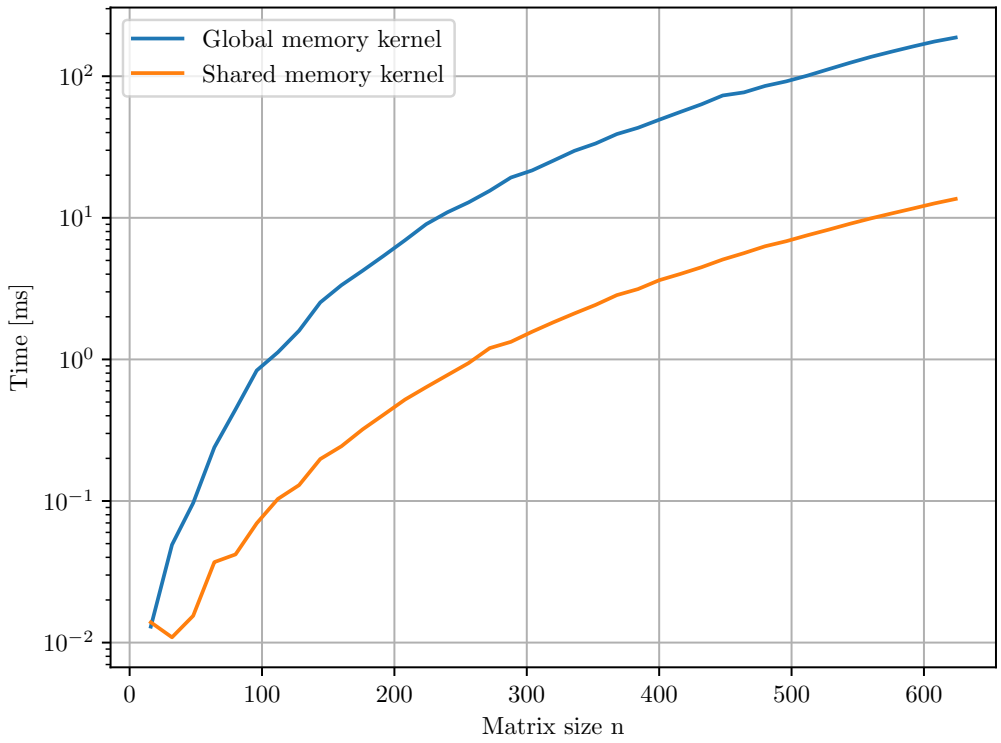

Figure 4.2: Single matrix-matrix parallel multiplication with global-memory utilisation (listing 4.3) and for shared-memory utilisation (listing 4.6). The matrices have dimensions $n \times n$. The calculations were done on a Nvidia GeForce 940MX GPU using single-precision floating-point operations.

shared memory. To summarise, parallel computing is central in GPU acceleration, and shared-memory utilisation is central in GPU optimisation. To use an algorithm efficiently on the GPU, it will have to be inherently parallel. However, if the algorithm relies heavily on memory transfer, it is vital that it is possible to divide it into chunks that fit in shared memory. Otherwise, all efficiency gained by parallel computing will be lost in memory transfer time.

Chapter 5

Numerical Study of Method Performance

In this chapter I will numerically investigate the performance of the WPCD method in terms of accuracy, precision, speed, and efficiency. I have written a parallel, GPU-accelerated implementation of the WPCD method, and as a benchmark I will use an optimised CPU implementation [26] of the matrix inversion (MI) method [29] presented in section 2.2.
The analysis is done in three parts, starting with the accuracy and precision of WPCD in section 5.2. Thereafter in section 5.3 I compare the speed of the WPCD and MI implementations, followed by a FLOPS efficiency comparison of the two implementations.

5.1 Calculation parameters

I collectively use the word "results" to mean calculated values for the total and differential cross sections, as well as for phase shifts in the Stapp convention (section 2.1.2). All calculations shown in this chapter were done for proton-neutron scattering using an optimised next-to-next-to-leading order chiral potential [69] called N2LO_{opt}. All observables, unless otherwise stated, were calculated using a partial-wave expansion with truncation at total angular momentum $J \leq 30$ (see equation 2.15). The WPCD results were calculated using momentum wave-packets (see equation 3.14) with boundaries set by a Chebyshev distribution with scaling factor $\alpha = 100$ MeV (see equation 3.36), and with an energy-averaged resolvent (see equation 3.34). The benchmark used, referred to as "exact", is a MI calculation with $N_{\text{GL}} = 96$ Gauss-Legendre points calculated at laboratory scattering energies

$T_{\text{lab}} = 10^{-5}, 1, 2, 3, \ldots, 350$ MeV. The calculations were done on the Vera cluster at the C3SE centre for computing [59].

5.2 Method accuracy and precision

The accuracy and precision of the WPCD method should be ascertained by the convergence towards the exact results with increasing wave-packet basis size N_{WP}. Ideally, in the $N_{\text{WP}} \to \infty$ limit we will approach a continuous basis such that the exact and WPCD results coincide regardless of the wave-packet distribution. In practice I will only investigate $N_{\text{WP}} \to \mathcal{O}(10^2)$. This makes it important to consider the three major approximations we do in WPCD when we want to separately determine the method accuracy and precision.
Firstly, the EQ weights were approximated to be proportional to the wave-packet bin widths as shown in equation 3.13. Secondly, we approximated L_2-integrals using EQ (equation 3.7), the quality of which we ultimately can only control through the wave-packet width/boundary distribution given the first approximation. Thirdly, the averaging of continuous states into wave packets means we use momentum-averaged matrix representations of operators in the LS equation. This averaging should improve with reduced bin widths and therefore by increasing N_{WP}. In short, we can only control the second and third approximations, and these will therefore determine the accuracy and precision of WPCD.
This makes it challenging to define a controlled procedure to separately test the precision and accuracy for the WPCD method. The parameters of the WPCD method should affect both accuracy and precision. However, since I will not investigate any distribution beyond the Chebyshev distribution, I will assume an increasing accuracy to be characterised by a systematic convergence onto the exact results with increasing N_{WP}. I will assume good precision to be similar behaviour with changing N_{WP}.

I start with an analysis of the WPCD prediction of phase shifts and mixing angles of partial-wave states (equation 2.15). I calculate them using the continuous T-matrix representation in equation 3.30 calculated at bin mid-point momenta $q' = q = \frac{q_{i+1}+q_i}{2}$, where $\{q_i\}_{i=0}^{N_{\text{WP}}}$ are the momentum bin boundaries, since these points are eigendifferential momenta/energies as shown in equation 3.10. In figure 5.1 I show the phase shifts for the partial waves $^1\text{S}_0$, $^3\text{P}_0$, $^3\text{S}_1$, and the mixing parameter $\epsilon_{^3\text{S}_1-^3\text{D}_1}$. As expected, the WPCD results for the phase shifts and mixing angles have one value in each bin due to energy averaging, giving a step-like trend that closely follows the exact curve. The bin widths appear to contract onto the exact curve in going from $N_{\text{WP}} = 32$ to $N_{\text{WP}} = 64$ for the phase shifts, signifying

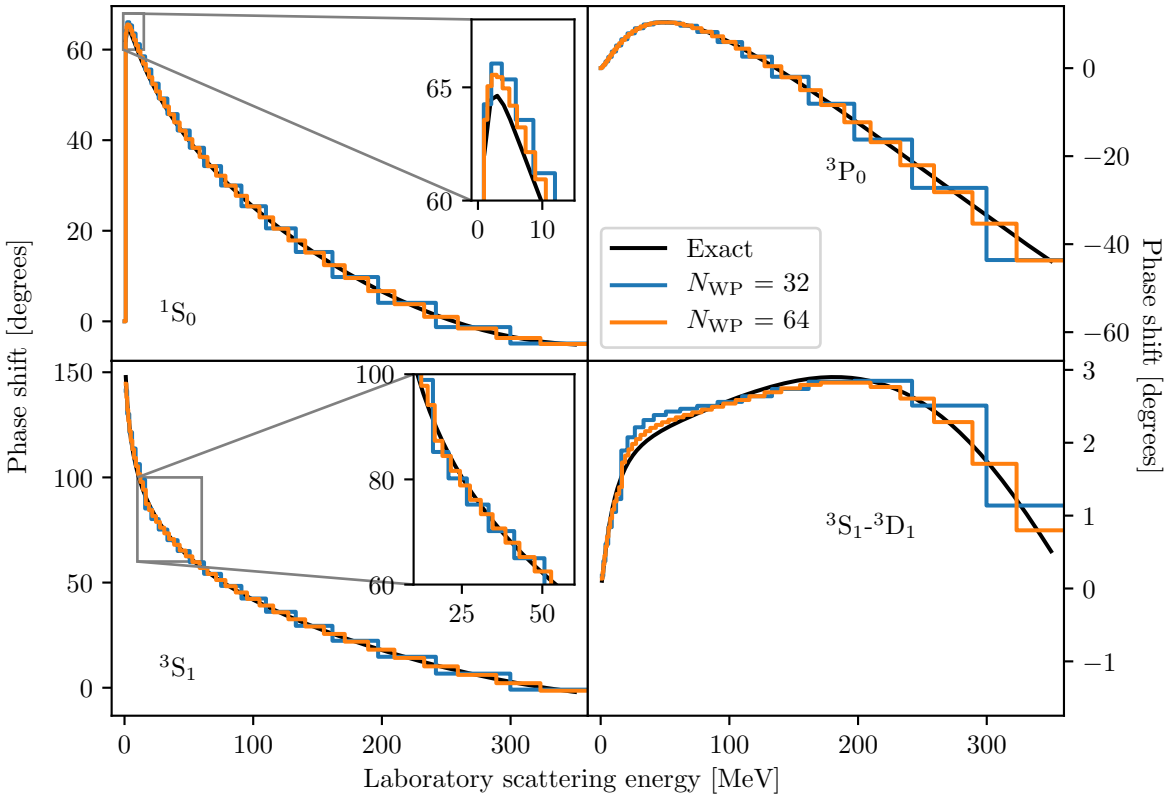

Figure 5.1: Phase shifts and mixing angle calculated with the WPCD method for N_{WP}=32 (blue) and N_{WP}=64 (orange) wave packets compared to exact results (black). The step-like trend is due to energy-averaging of the resolvent.

increasing precision. The clear trend in the mixing angle seen as the curve moves closer to the exact curve is a good sign at increasing accuracy with basis size. However, there are three trends here that will reappear in later figures, and therefore deserve more detailed comments:
Firstly, WPCD results for $^1S_0$ consistently overestimate the exact result, especially near the peak of the phase shift. This overestimation might be the major cause of cross section deviations at very low energies for WPCD.
Secondly, the WPCD results for $^3S_1$ cross the exact results at laboratory energy $T_{lab} \sim 25$ MeV and again at $T_{lab} \sim 35$ MeV. This trend is consistent at all basis sizes and is due to the treatment of the deuteron bound state in WPCD. Calculating a phase shift δ is done via inverse trigonometric functions giving $\delta \in [-90, 90]$ degrees.

A bound state is characterised as a transition $\delta(E+\epsilon)-\delta(E) = 180$ degrees at some energy E for an infinitesimal positive step ϵ, due to Levinson's theorem [70]. It is apparent that this transition is difficult to mimic for the WPCD method at $T_{\text{lab}} \sim 25$ MeV for $^3\text{S}_1$. The second crossing at $T_{\text{lab}} \sim 35$ MeV is due to the curve returning to the consistent overestimation of the $^3\text{S}_1$ phase-shift.
Thirdly, the $\epsilon_{^3\text{S}_1-^3\text{D}_1}$ mixing angle in WPCD shows a much clearer deviation from the exact result in both the low ($T_{\text{lab}} < 100$ MeV) and high ($T_{\text{lab}} > 250$ MeV) energy regions. However, this deviation is believed not to be a major source of error in observable calculations, in comparison to phase-shift deviations. A detailed sensitivity analysis could provide more insight. The deviation is likely related to the bound state inaccuracy already discussed.
Partial waves with higher angular momenta than shown here will contribute less due to the limitations imposed by the centrifugal barrier, in the energy region in the figure.

As mentioned, one of the approximations we make in the WPCD method is the momentum-averaging of operators. This is a possible source for the overestimate seen in the $^1\text{S}_0$ peak. In figure 5.2 we see the continuous potential matrix $\langle q|\hat{v}|q'\rangle$ in the $^1\text{S}_0$ channel, using a $N_{\text{WP}} = 32$ wave-packet Chebyshev distributed grid overlay, as an illustration of momentum-averaging. Within each square the wave-packet matrix will have a single value according to equation 3.31. An indication that the wave-packet $^1\text{S}_0$ potential matrix at small basis sizes ($N_{\text{WP}} \leq 64$) may have too low resolution to reproduce the peak is revealed by the observed colour variation within the high-momentum bins. While this coarse-graininess effect on the peak is important to keep in mind for small bases, it is suspected to quickly become an insignificant source of error with increasing N_{WP}.

In the left part of figure 5.3 I show the total cross section calculated using the WPCD method for $N_{\text{WP}} = 32$ and $N_{\text{WP}} = 64$. In the right part of the figure I show the absolute values of the corresponding relative difference between the exact and WPCD curves. While it is clear that the difference decreases systematically for increasing N_{WP}, it is very erratic, changing rapidly in a region from $\mathcal{O}\left(10^{-2}\right)$ mb to $\mathcal{O}\left(10^{3}\right)$ mb. The sawtooth-like trend is well understood as due to the step-like curves given by WPCD, but this nonetheless makes it challenging to employ averaged values of observables, in particular for method precision quantification. Therefore, a scheme is required to smooth the curves of the phase shifts (and thereby cross sections) and consequently to smooth the relative difference.

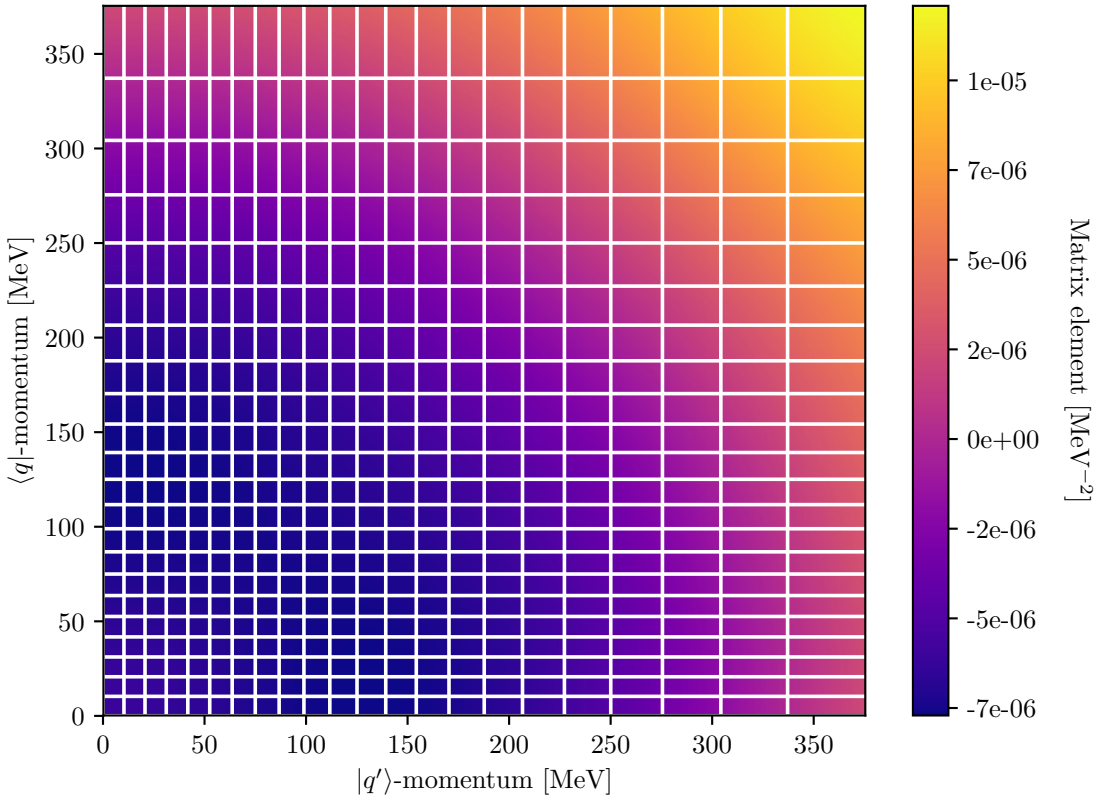

Figure 5.2: $^1S_0$ potential matrix in continuous representation with a $N_{\text{WP}} = 32$ wave-packet Chebyshev distributed grid overlay.

5.2.1 Linearly interpolating phase shifts

To treat the effects due to bin-averaging, a linear interpolation scheme has been used. As mentioned, I calculated phase shifts using the continuous T-matrix elements evaluated at momentum bin mid-points. The motivation for using mid-points is due to the eigendifferential energy in equation 3.10; the wave-packet eigenvalue is the bin mid-point. To calculate observables for any scattering energy, we should devise a scheme to interpolate phase shifts such that we have a smooth curve rather than the step-like curve seen in figure 5.1. One possibility is to simply use the discrete and continuous T-matrix relation (equation 3.30) at all momenta. Using momentum wave-packets, this equation simplifies to,

$$T(q,q) \approx \frac{1}{d_i q^2} \bar{T}_{ii}(E) \,. \tag{5.1}$$

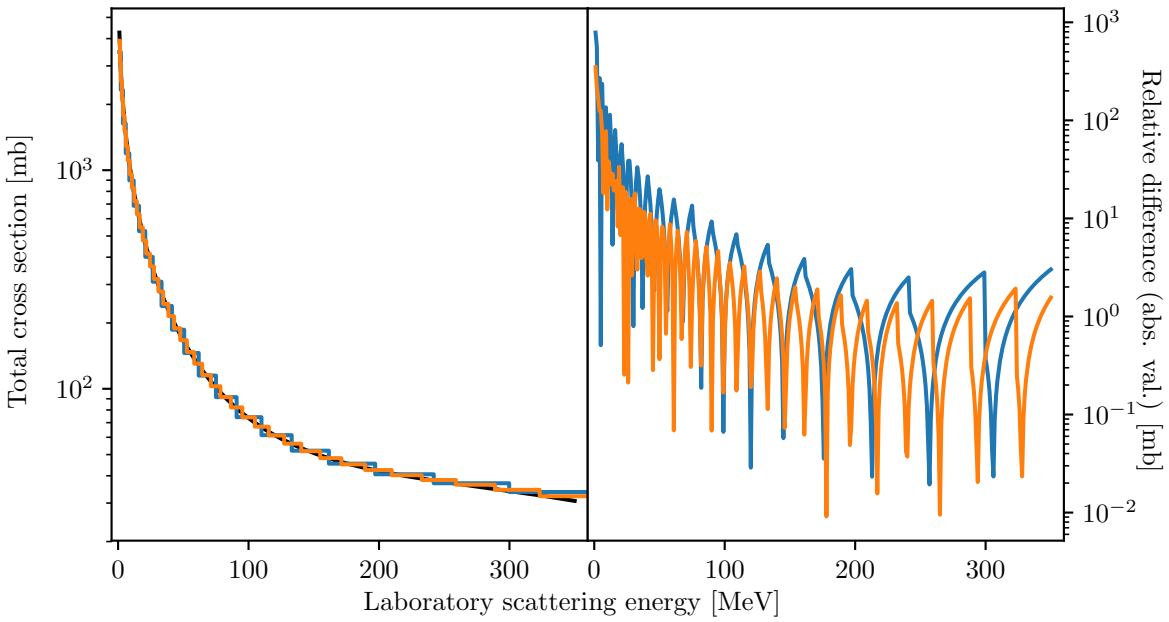

Figure 5.3: Total cross section (left) and the corresponding absolute values of the relative difference (right) calculated with the WPCD method. The colour coding is the same as in figure 5.1.

The coefficient $\frac{1}{d_i q^2}$ will decrease with increasing momentum. This equation will therefore simply turn all the steps in figure 5.1 into second order polynomial curves. Furthermore, it would not be smooth.

A better method to smooth the curve is by interpolation. I have chosen to focus on a simple linear interpolation. The mid-points $\bar{q}_i = \frac{1}{2}(q_{i-1} + q_i)$ of each bin in figure 5.1 appear very close to the exact curve, as we might expect from the wave-packet eigenvalues. I used this observation to calculate phase shifts using the continuous $T(\bar{q}_i, \bar{q}_i)$-elements, and linearly interpolating the phase shifts $\delta_i \equiv \delta(\bar{q}_i)$ by

$$\delta(q) = \left(\frac{\delta_i - \delta_{i-1}}{q_i - q_{i-1}}\right) q + \left(\delta_{i-1} - \frac{\delta_i - \delta_{i-1}}{q_i - q_{i-1}} q_{i-1}\right) . \tag{5.2}$$

Of course, the phase shifts can be linearly interpolated using other points besides the mid-points in equation 5.1. For example, we can let q be

$$q = q_{i-1} + \frac{n}{m}(q_i - q_{i-1}) , \tag{5.3}$$

for some $n \in [0, m]$, such that $n = \frac{m}{2}$ is the bin mid-point again. However, I stress that the mid-point interpolation is the best motivated choice due to the wave-packet eigenvalues.

In figure 5.4 I show the interpolation of figure 5.1 using equation 5.2. The bands show the phase-shift variation as a function of interpolation point. The bands, referred to henceforth as variation bands, span the resulting $\delta(q)$ calculated with $m = 10$ and $n = [0.1, 1, 2, \ldots 8, 9, 9.9]$ in equation 5.3. We see that mid-point interpolation is very close to the exact curve in comparison to the band widths. Furthermore, we see that the bands grow narrower with larger wave-packet basis; a good sign of convergence with increasing basis size.
More importantly, however, in the lower part of the figure I show the interpolated total cross section and the corresponding absolute difference between the exact and WPCD interpolated curves. The difference shows that the erratic behaviour in figure 5.3 is now smoothed out and remains stable with a significantly lower relative difference. There is still some erratic behaviour in the low energy region ($T_{\text{lab}} < 40$ MeV), but this is believed to originate from the aforementioned trends in the $^1\text{S}_0$ and $^3\text{S}_1$ partial waves.

Lastly, in figure 5.5 I show the differential cross section for all scattering angles θ at increasing laboratory energies T_{lab}. It is noticeable that the widths of the variance bands depend on the energy. The narrowest stretch of the bands tends to the right with increasing T_{lab}. This trend is not yet fully understood, but a promising explanation is that low angular momentum partial waves become less influential on the cross section as T_{lab} grows, including deviations from the exact results. Therefore, due to the relation between partial waves and the scattering angle dependence in the spherical harmonics in the PWE (equation 2.15), WPCD "errors" will either mitigate or shift with respect to the scattering angle θ and energy T_{lab}.

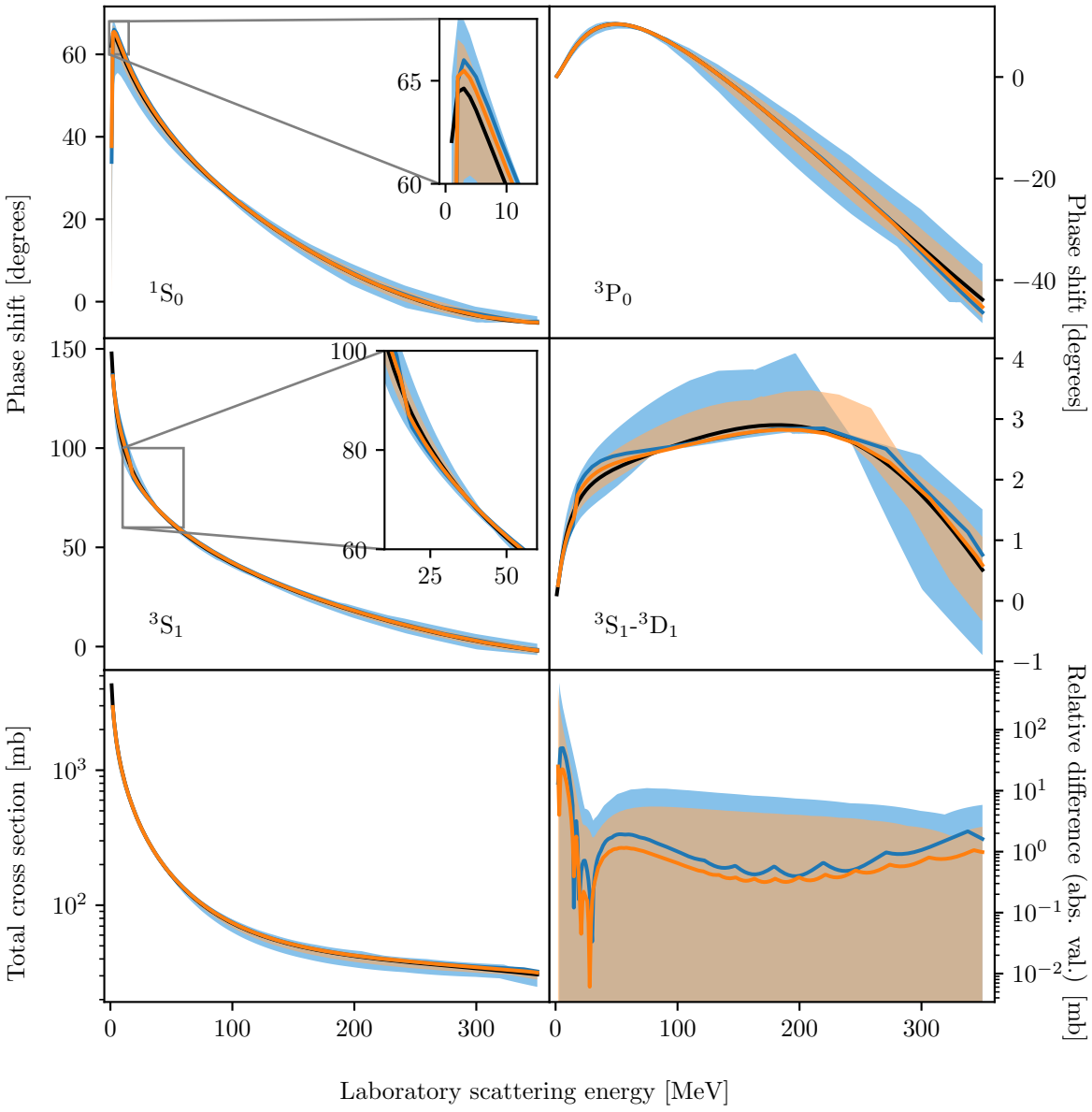

Figure 5.4: Phase shifts for the $^1S_0$, $^3P_0$, and $^3S_1$ partial waves and the $^3S_1$-$^3D_1$ mixing angle, calculated using WPCD. The colour coding is the same as in figure 5.1. The variation bands show the corresponding interpolation variance, using the same colour coding with $N_{\mathrm{WP}} = 32$ (light blue) and $N_{\mathrm{WP}} = 64$ (light orange). The bottom row shows the total cross section with corresponding absolute values of the relative difference between exact and WPCD curves.

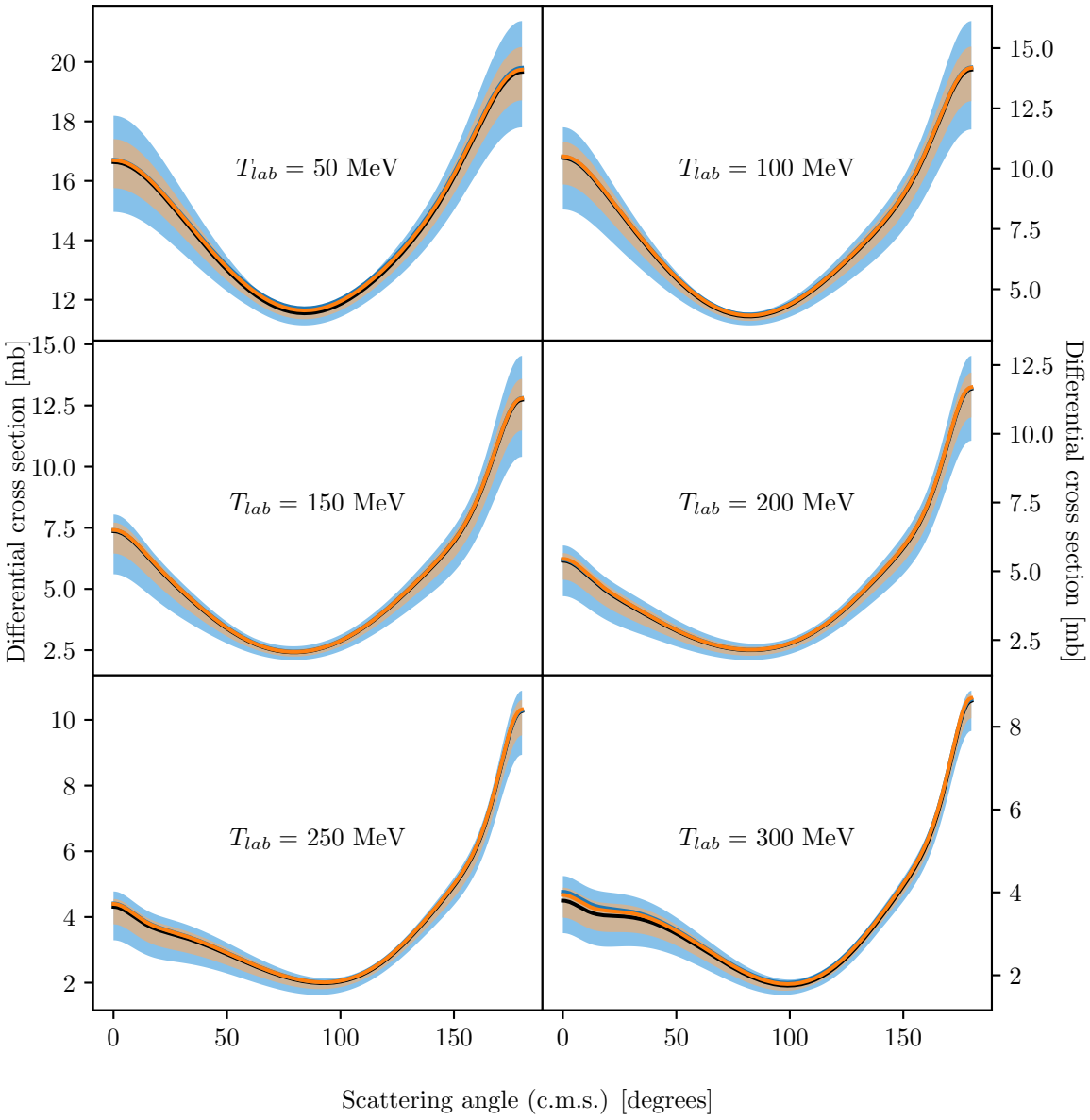

Figure 5.5: Differential cross sections calculated using WPCD. The colour coding is the same as in figure 5.4.

5.2.2 Relative deviation with large bases

The WPCD results studied so far have been limited to fairly small wave-packet bases $N_{\mathrm{WP}} \leq 64$. While it is arguably unnecessary to go beyond $N_{\mathrm{WP}} = 64$ to get insight into the trends seen so far, it is interesting to see if the relative deviation from the exact results converges fully at some $N_{\mathrm{WP}} > 64$.
In figure 5.6 I show a heat map of the total cross-section relative error for a large range of wave-packet basis sizes. The error changes little beyond $N_{\mathrm{WP}} = 10^2$ wave packets. There are three prominent dark lines where the MI and WPCD methods accidentally agree. Two of these are due to exact curve-crossing seen in the $^3\mathrm{S}_1$ state, as discussed earlier. From the heat map it can be concluded that this trend of curve-crossing for $^3\mathrm{S}_1$ happens for all basis sizes $N_{\mathrm{WP}} \leq 500$. In the very low energy region $T_{\mathrm{lab}} < 10$ MeV there is a significantly larger difference between WPCD and the exact results. The $^1\mathrm{S}_0$ phase-shift is the dominant term in the PWE and believed to be the major cause of this. It is apparent that the WPCD method converges quite quickly with a Chebyshev distribution, and it is typically not necessary to go beyond $N_{\mathrm{WP}} = 10^2$.

To summarise briefly the most important observations made in the WPCD study so far: The use of momentum-averaged phase-shifts introduces highly erratic behaviours in the total cross sections. To remedy this, I introduced a linear interpolation scheme using bin mid-points on the phase shifts in order to smooth the phase shifts, observables, and deviation of observables. The total cross section deviation does not change significantly beyond $N_{\mathrm{WP}} > 10^2$.
A conclusion based on these findings is that the smooth deviations seen after linear interpolation motivates the use of the total cross section root-mean-square error (RMSE) as a collective measure of method precision and accuracy.

5.2.3 Method RMSE comparison

I define the RMSE for total cross sections by,

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} \left(\sigma_{\mathrm{exact},i} - \sigma_{\mathrm{method},i}\right)^2}{n}}\,, \tag{5.4}$$

where $\sigma_{\mathrm{exact},i}$ and $\sigma_{\mathrm{method},i}$ denote, respectively, the numerically exact and WPCD- or MI-calculated total cross sections at some scattering energy E_i for $i = 1, \ldots, n$. By using the RMSE of total cross sections I can compare the WPCD and MI methods, reserving $N_{\mathrm{WP}} = 10^2$ as a converged limit for the WPCD method. While the method

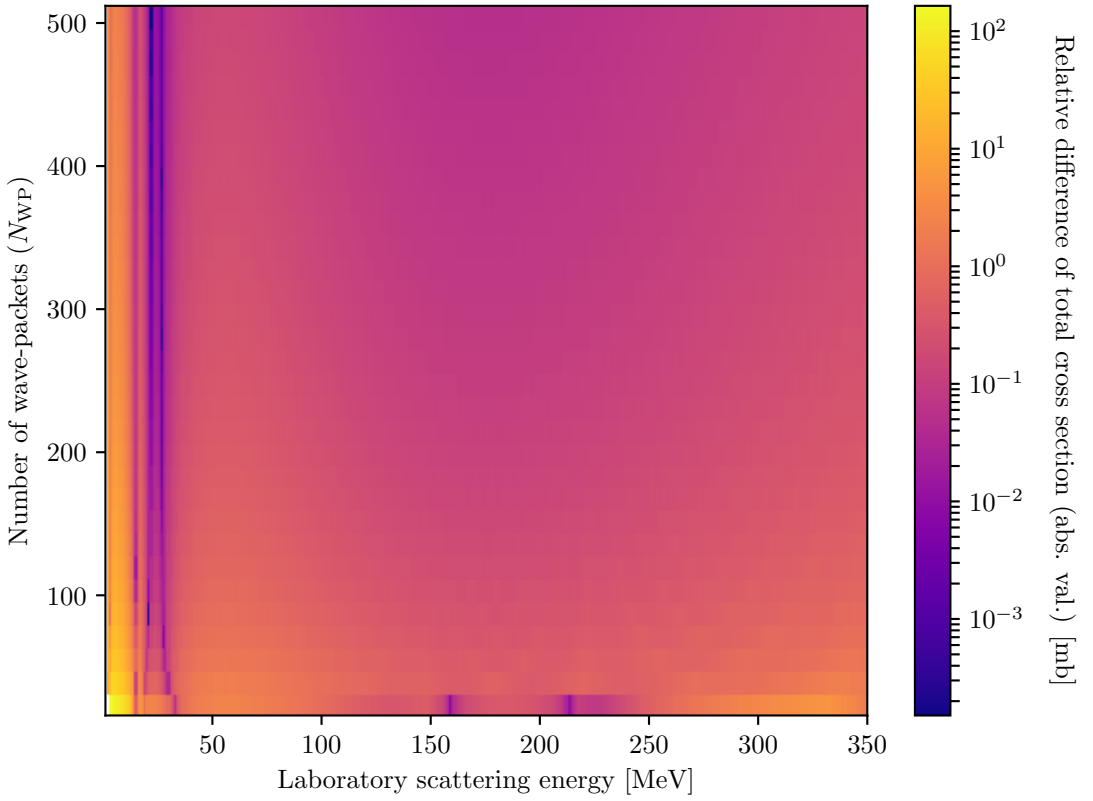

Figure 5.6: Absolute values of the relative difference between exact results and WPCD method for the total cross section, shown as a function of wave-packet basis size N_{WP} and laboratory energy T_{lab}.

RMSE is interesting in and by itself, it will ultimately serve as a tool to compare time profiles at given RMSEs.
Besides method error and RMSE, an important aspect of NN scattering calculations is the truncation of the PWE (equation 2.15) at some maximum total angular momentum $J \leq J_{\mathrm{max}}$. In all previous figures I showed results with a large value $J_{\mathrm{max}} = 30$. As mentioned, however, the partial-waves contribute less and less for increasing angular momenta. If the method RMSE converges already at a lower truncation value (e.g. $J_{\mathrm{max}} = 10$), it would imply we are wasting a significant amount of computing time on phase-shift calculations that are insignificant for observables.
In figure 5.7 I show the RMSE for $T_{\mathrm{lab}} \in (0, 350]$ MeV as a function of J_{max} for both the MI and WPCD method. The WPCD results are for cases where $N_{\mathrm{WP}} < 10^2$ due to the convergence we have seen above this region from figure 5.6. While the

WPCD results use interpolation of bin mid-points, the MI results are calculated at each scattering energy of interest as stated in 5.1.
From this figure it is clear the MI and WPCD methods do not have comparable RMSE at equal bases sizes $N_{\mathrm{WP}} = N_{\mathrm{GL}}$. Instead we see that the MI method at $N_{\mathrm{GL}} \leq 16$ reaches similar convergence as the WPCD method for $N_{\mathrm{WP}} \geq 64$, and that MI improves exponentially as N_{GL} grows. Furthermore, due to the large RMSE induced by the low energy deviation for the WPCD method, the WPCD results converge already at $J_{\max} \approx 4$. For the MI method we can improve the RMSE down to machine precision, although at $N_{\mathrm{GL}} = 96$ we are simply recalculating the exact results.
Without a time-profile it is impossible to say whether WPCD at e.g. $N_{\mathrm{WP}} = 96$ can perform faster than MI at[1] $N_{\mathrm{GL}} = 16$, which is the ultimate purpose here. Nonetheless, it is interesting to see if the WPCD RMSE can be in the energy region above $T_{\mathrm{lab}} \geq 40$ MeV.

The erratic deviations of the WCPD method occur below T_{lab}= 40 MeV, which corresponds to a c.m.s. momentum that is very close to the pion mass, $q \approx m_\pi =$ 138.039 MeV. To further investigate the potential of the WPCD method, the RMSE can be calculated in a restricted $q > m_\pi$ region.
In figure 5.8 I show this RMSE in the same type of format as in figure 5.7. It appears that both the MI and WPCD results yield RMSE values around 10 mb for low N_{GL} and N_{WP}. However, we see that $N_{\mathrm{WP}} = 16$ gives an RMSE at around 2.0 mb which is noteworthy for two reasons:

- The WPCD coupled channel Hamiltonian will be of size $2N_{\mathrm{WP}} \times 2N_{\mathrm{WP}}$ (see section 3.4), meaning we diagonalise 32×32 matrices. These matrices fit entirely on the GPU shared memory, allowing for a strong reduction in GPU memory read/write demand while diagonalising Hamiltonians.

- A recent Bayesian uncertainty quantification analysis of chiral interactions suggests that a next-to-next-to-leading order (NNLO) χEFT NN model error in the $q > m_\pi$ region around is at least 2 mb at 68% degree-of-belief (DoB) and at least 5 mb at 95% DoB [25].

A more model-specific reason why the $q > m_\pi$ region is of interest is because the expansion parameter in χEFT is partly defined by the pion mass. The expansion parameter is Q/Λ [71], where Q is the soft scale and Λ is the hard scale of the system

[1]These two basis sizes appear to give comparable RMSE, altough convergent at different $J_{\max}$.

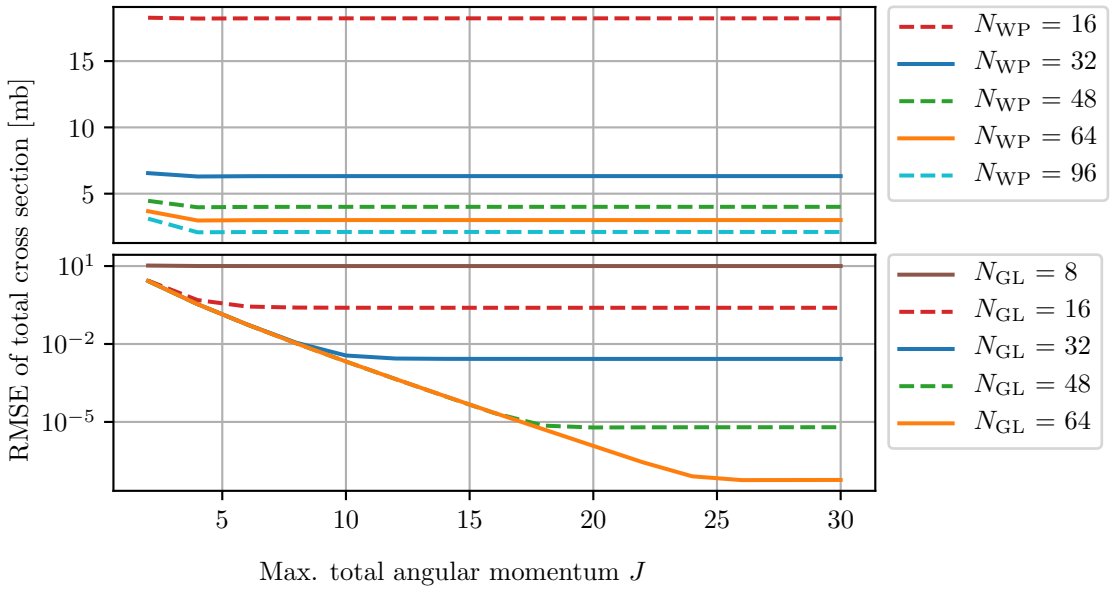

Figure 5.7: Root-mean-square error (RMSE) of the total cross section for $T_{\mathrm{lab}} \in (0, 350]$ MeV as a function of maximum total angular momentum J_{max} in the PWE, for both the number of grid points N_{GL} in the MI method and the number of wave packets N_{WP} in the WPCD method. WPCD results are calculated with mid-point interpolation.

under study. The value of Λ is fixed, and Q is given by

$$Q = \max(m_\pi, q) \, . \tag{5.5}$$

Based on the improved RMSE in the $q > m_\pi$-region it may be that the WPCD method is particularly well-suited for Bayesian studies in a kinematic region $q > m_\pi$.

The RMSE comparison so far has been made by letting the MI method calculate observables at every scattering energy given in section 5.1, which as we know from section 2.2.1 increases the complexity linearly. However, there is nothing to prevent a linear interpolation of MI phase-shifts as I do with the WPCD method. Furthermore, by choosing the MI interpolation points equal to the WPCD interpolation points, i.e. the bin mid-points from a Chebyshev distribution of wave packets, we can examine the optimal RMSE possible when interpolating phase shifts due to the precision of the MI method.
In figure 5.9 the RMSE of the MI method is shown as a function of the number

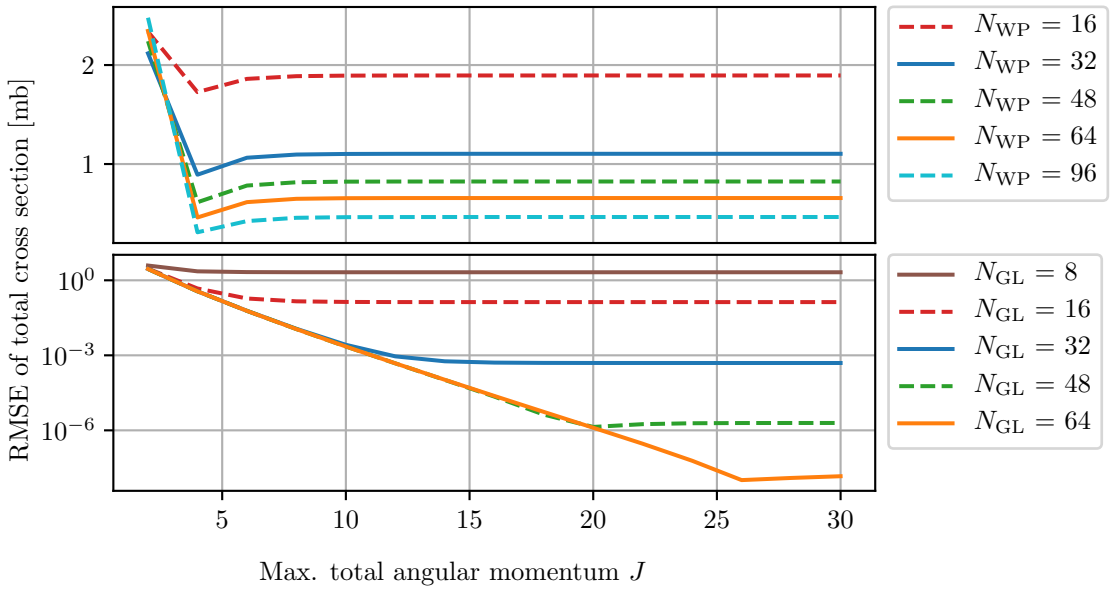

Figure 5.8: Root-mean-square error (RMSE) of the total cross section for $T_{\mathrm{lab}} \in [40, 350]$ MeV as a function of maximum total angular momentum J_{max} in the PWE, for both the number of grid points N_{GL} in the MI method and the number of wave packets N_{WP} in the WPCD method. WPCD results are calculated with mid-point interpolation.

of interpolation points, for the whole energy range and the $q > m_\pi$ range. As a benchmark I include the results from the WPCD method.
In the full energy region, the RMSE of the MI method with $N_{\mathrm{GL}} = 8$ is largely unusable. The figure shows that for $N_{\mathrm{GL}} \geq 16$ the RMSE is dominated by the limitations of linear interpolation in a Chebyshev distribution. This means that the discrepancy between the $N_{\mathrm{GL}} = 16$ and WPCD line is due to the limitations of the WPCD method, ruling out suspicions of possible limitations of linear interpolation for WPCD.
For the $q > m_\pi$ region the conclusion is quite similar. The MI method shows a clearer signature of convergence at $N_{\mathrm{GL}} = 32$ rather than $N_{\mathrm{GL}} = 16$ as before. The discrepancy between WPCD and MI for $N_{\mathrm{GL}} = 32$ lies in the 0.4-1.0 mb region.
A side note on the MI results shown here is that they can possibly be improved further by a better distribution of interpolation points.

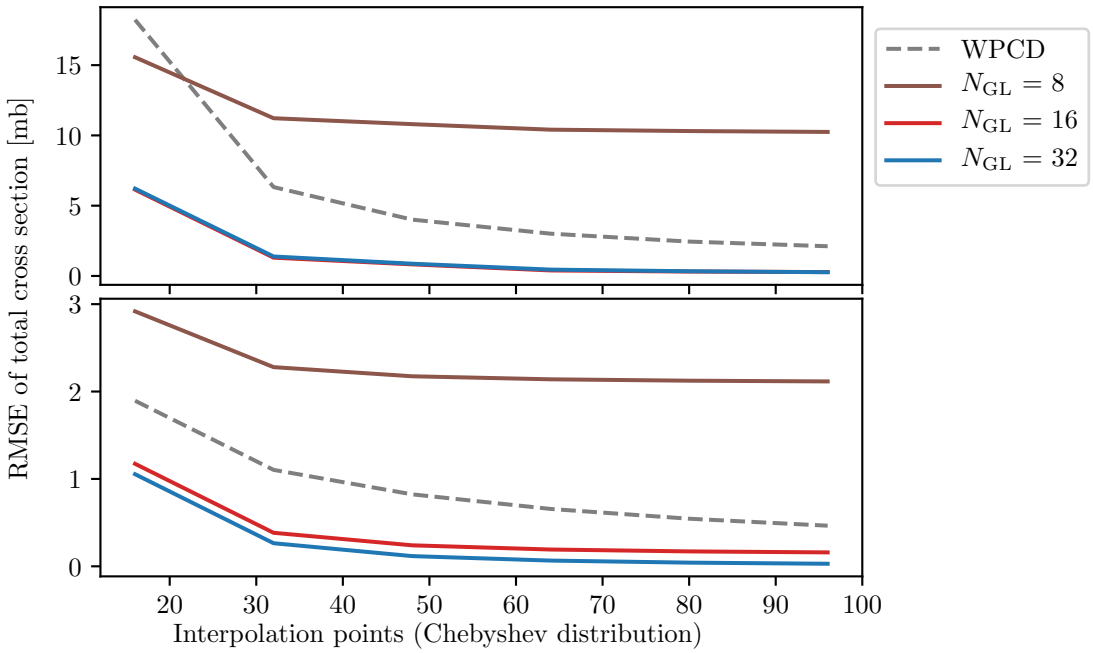

Figure 5.9: Root-mean-square error (RMSE) of the total cross section as a function of Chebyshev distributed interpolation points for the MI method (solid curves) with the WPCD method as benchmark (dashed curve). The top panel includes the whole energy range in the RMSE and the bottom panel includes just the $q > m_\pi$ region). Note that the $N_{\mathrm{GL}} = 16$ and $N_{\mathrm{GL}} = 32$ results overlap in the top panel.

5.3 Time profiling and program efficiency

The WPCD implementation has been written with GPU acceleration, utilising the CUDA interface with the cuBLAS[72] and cuSOLVER[73] libraries for linear operations. The time profile comparison of the two implementations is shown in the top of figure 5.10, using interpolation for both MI and WPCD. The WPCD method is faster than the MI method at every choice for $N_{\mathrm{WP}} = N_{\mathrm{GL}}$, but we know that the RMSE is not equal between the two methods when $N_{\mathrm{WP}} = N_{\mathrm{GL}}$. Using this time profile, I use the RMSE values shown in figure 5.9 in the $q > m_\pi$ region to generate table 5.1. The table shows the time each method uses to achieve a given RMSE.
From table 5.1 it can be seen that the WPCD method performs slightly better (1-2 times faster) in the $q > m_\pi$ region for every RMSE. The exception is for RMSE equal to 2.0 mb, where the WPCD method is nearly 30 times as fast. This could be due to the significant reduction in GPU global memory read/write usage by fitting entire

Hamiltonian matrices onto shared memory.
It is important to analyse the efficiency of the two implementations. Only looking at computing times can give a wrong impression of the capabilities of the implemented methods.

Table 5.1: Time used by WPCD and MI (interpolated from Chebyshev distribution) for different RMSEs in the $q > m_\pi$ energy region, based on figures 5.9 and 5.10. Note in the top two rows of the WPCD section there are no corresponding RMSE in figure 5.9 and therefore these values are not applicable.

RMSE [mb]	MI			WPCD	
	N_{GL}	n_E	Time [ms]	N_{WP}	Time [ms]
3.0	8	16	5	N.A.	N.A.
2.5	8	32	11	N.A.	N.A.
2.0	8	48	~ 15	16	0.5
1.5	16	16	11	16-32	1-5
1.0	16	24	~ 8	32	6
0.5	16	32	20	64	12

5.3.1 Efficiency of implementation

Measuring implementation efficiency of a method can be difficult if done very thoroughly. We must take into account the sequential ordering of the different parts of the algorithm, the memory transfer times in algorithms, etc. For the efficiency tests done here I will use the ratio of CPU or GPU FLOPS divided by the MI or WPCD complexity, respectively. This ratio will represent the theoretical limit for the minimal time a processor can use on solving the method. The measured times divided by the theoretical limit will give an indication of the implementation efficiency. Note that it is likely to be very low since the complexity models presented in sections 2.2.1 and 3.6 did not consider any ordering of the steps, e.g. diagonalising the Hamiltonian matrix before solving the LS equation.
The theoretical upper limit on FLOPS is hardware-specific, and can be calculated as

$$\begin{aligned} \text{FLOPS} = \quad & (\text{Clock frequency}) \\ & \times (\text{Number of cores}) \\ & \times (\text{Instructions per cycle}) \\ & \times (\text{Operations per instruction}) \,. \end{aligned} \tag{5.6}$$

In the lower part of figure 5.10 I show the calculated efficiencies based on equation 5.6 and table 4.1. It is apparent that the two implementations have comparable efficiencies.
The MI efficiency does not appear to change with respect to N_E, as we would expect from the linear energy scaling in the MI complexity model. The alarmingly low ef-

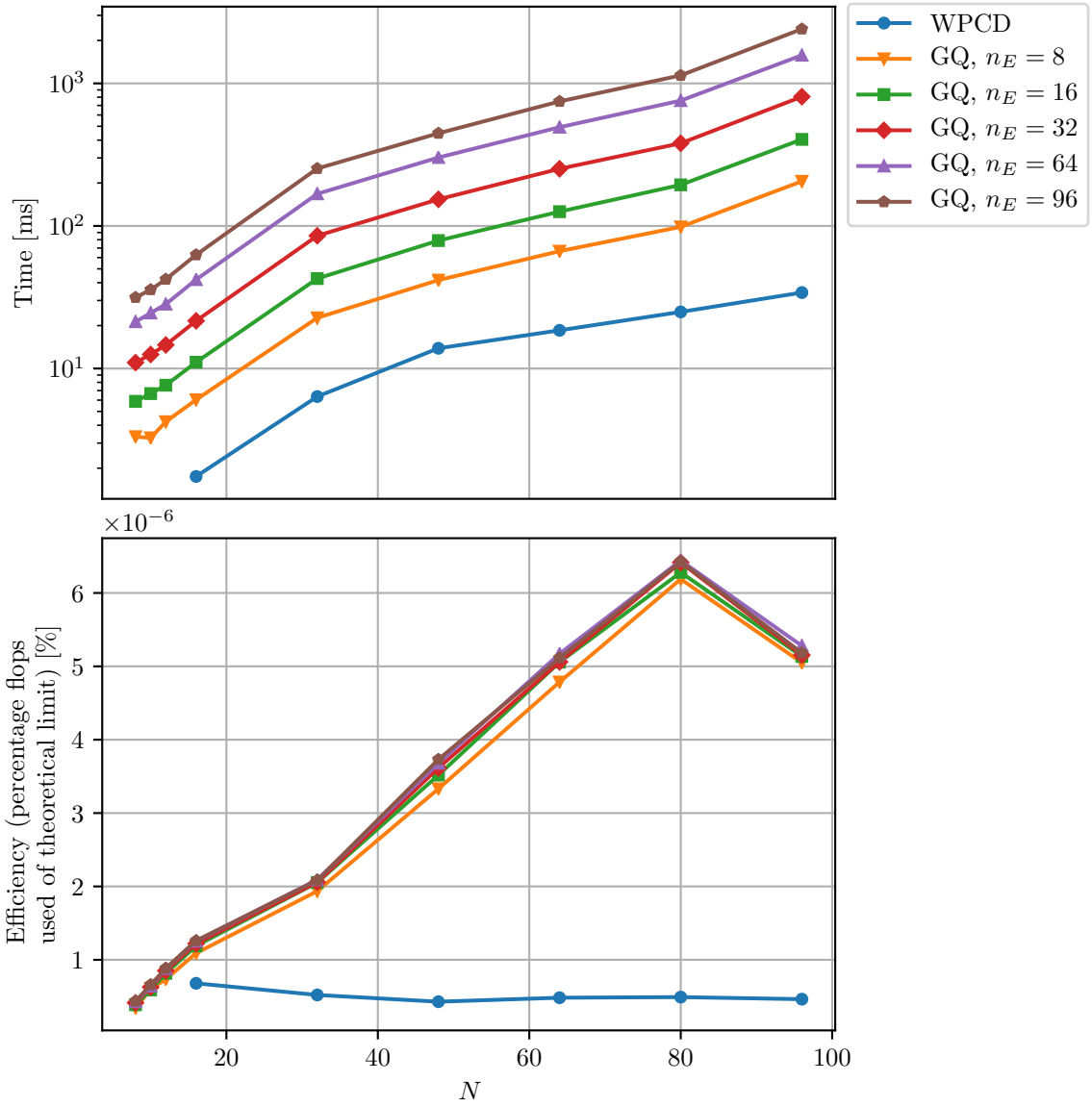

Figure 5.10: Top: Time to solve the LS equation and calculate phase shifts for $J_{\max} = 30$ using the WPCD (blue) and MI (remaining) methods. Here, $N = N_{\text{WP}}$ for the WPCD method and $N = N_{\text{GL}}$ for the MI method. The MI results have been calculated for N_E on-shell energies. Bottom: Corresponding program efficiency, using hardware specifications (table 4.1) for the Intel Xeon Gold 6130 [66] CPU (for MI results) and the Nvidia Tesla V100 [63] GPU (for WPCD results), both stationed at the Vera cluster at C3SE [59]. The MI results were calculated using double floating-point precision, while the WPCD results used single precision.

ficiency ($\mathcal{O}(10^{-5})$ percent) is most likely due to both memory transfer time and the sequential parts of the algorithms.
Note that the figure does not tell us anything about the room for optimisation. For such an investigation we would have to profile separate linear algorithms, like diagonalisation, and study hardware bandwidth and transfer rates to get a better picture of efficiency. This falls outside the scope of the project.

An important last comment regards the GPU transfer times and the efficient use of shared memory. To exemplify the importance of efficient memory use, in figure 5.11 I show the time usage for three different GPUs and the decomposition of time usage for the Nvidia V100 GPU. We see that the majority of execution time is due to the diagonalisation of the Hamiltonian matrix. The memory accessing of the Jacobi method is significant, which is why we see such a clear difference between the three GPUs: V100, T4, and K40. These three GPUs have significant differences in memory technology [60].

5.4 Summary

In general, there appears to be a limited gain of speed in using the WPCD method as opposed to the MI method. I have found that the WPCD method implemented on a GPU is roughly a factor two faster than a highly optimised CPU MI implementation, with the exception of using a 16 wave-packet basis in which case entire Hamiltonian matrices can fit onto GPU shared memory and reduce global transfers significantly. The WPCD method performs best in the c.m.s. momentum region above the pion mass. This may be quite interesting for χEFT Bayesian studies using WPCD due to the form of the χEFT expansion parameter.
The significant increase in the RMSE for the WPCD method at momenta $q < m_\pi$ is due poor reproduction of the $^1S_0$ and $^3S_1$ partial-wave phase-shifts. Bound states and corresponding signatures in the phase shifts, due to Levinson's theorem, appear to be tricky for the WPCD handle. The MI method can give improving RMSEs with more interpolation points, and that this significantly affects calculation time. Furthermore, there is promise in trying to reduce the RMSE for MI by changing the distribution of interpolation points beyond the Chebyshev distribution.
The coarse graininess introduced by momentum averaging should quickly vanish above 32 wave packets, leaving the major source of method error to be the quality of the equivalent quadrature. This quality can only be tuned through the distribution of wave packets and the basis size. Finding a better distribution of wave packets beyond the Chebyshev distribution is a yet unexplored topic that is of high interest.

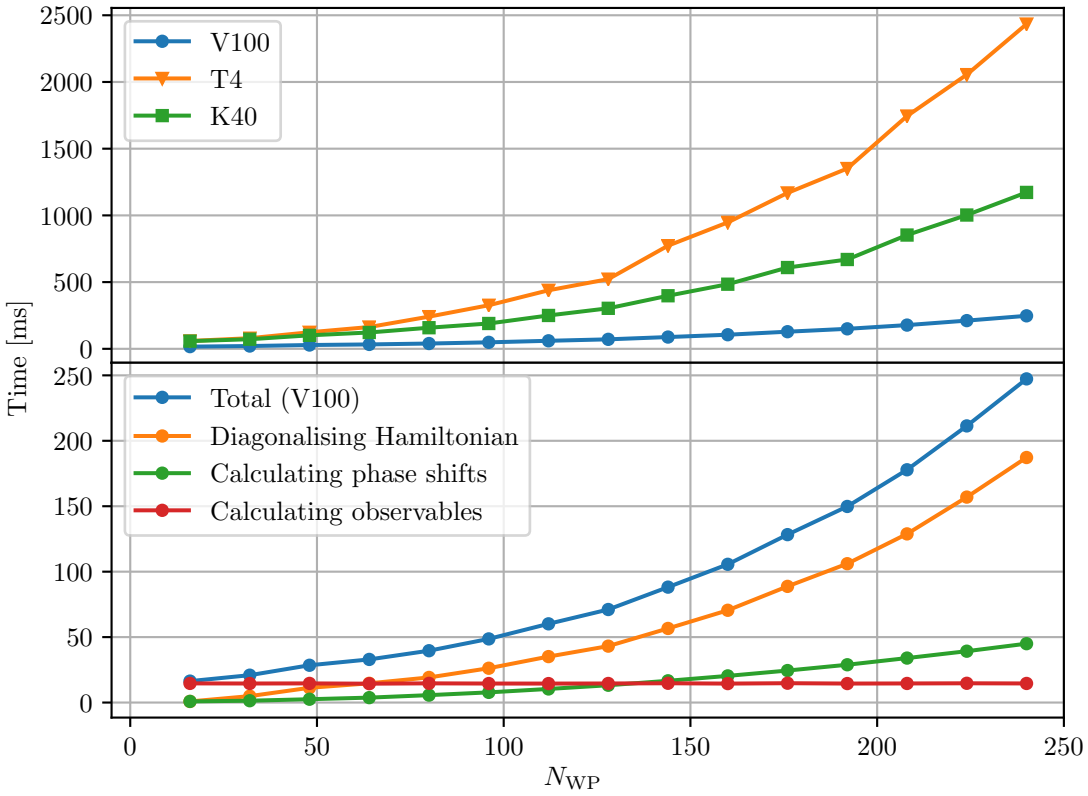

Figure 5.11: Top: total time spent solving the LS equation and calculating/interpolating phase shifts for three different Nvidia GPUs of the Tesla line; T4[62] (orange), V100[63] (blue), and K40[61] (green). Bottom: time decomposition of V100 total time, with focus on key parts of calculating NN scattering using the WPCD method.

Chapter 6

Conclusion and Outlook

In this thesis I have analysed the prospect of using the WPCD method [28] for speeding up NN elastic scattering calculations. This approach could be useful for e.g. likelihood evaluations in Bayesian parameter estimation and for uncertainty quantification of χEFT models of the strong nuclear force [74, 75]. The investigations were carried out using the chiral N2LO_{opt} interaction [69] applied to neutron-proton elastic scattering.
The standard approach for calculating NN scattering observables is based on a highly accurate and precise matrix inversion method [29]. However, the computational cost of matrix inversion at every experimentally relevant scattering energy is significant. The WPCD method [53, 55, 56] can circumvent this matrix-inversion step at the cost of a single eigendecomposition of the NN Hamiltonian represented in a wave-packet basis. Furthermore, the WPCD method is an inherently parallel algorithm and is therefore highly suitable for implementation on a GPU; a computer processor device capable of massive parallel throughput of computations albeit at a lower clock frequency compared to a CPU.

The WPCD method offers significantly higher computational efficiency for a basis size[1] n compared to the matrix inversion method. For a fixed basis size and n_E on-shell scattering energies, the number of arithmetic operations it takes to solve the LS equation is $\mathcal{O}\left(\frac{8}{3}n_E n^3\right)$ for the matrix inversion method, while the sequential and parallel WPCD methods use $\mathcal{O}\left(6n^3\right)$ and $\mathcal{O}\left((n+1)\log^2(n)\right)$ operations, respectively. Note the n_E-independence in the WPCD method. As expected, I find that the parallel WPCD algorithm will be highly dependent on the memory speed and

[1]The basis size n is the number of wave packets in WPCD, while in the matrix inversion method it is the number of quadrature points.

bandwidth of the parallel processing hardware to be used (CPU(s) or GPU(s)).
The matrix inversion is far more precise for equivalent basis sizes. There are most likely two reasons for the observed precision loss of the WPCD method. Firstly, the momentum-averaging of the NN potential matrix in a wave-packet basis necessarily leads to a coarse-grained representation. This introduces a noticeable effect in the calculation of the low energy $^1S_0$ phase-shift. Secondly, the approximation of the EQ weights depends on the wave-packet bin widths, as defined by the bin boundaries. In this work, the distribution of bin boundaries followed a Chebyshev distribution [46, 47, 49]. However, this is not necessarily the optimal choice for approximating scattering observables. The resulting method error in the calculation of the NN total cross section is around $10 - 10^2$ mb in the momentum region below the pion mass, $q < m_\pi$. Cross sections at scattering energies corresponding to momenta above m_π exhibit significantly lower error around 1 mb, uniformly up to laboratory scattering energies $T_{\text{lab}} = 350$ MeV. These errors were calculated with all partial waves up to and including total angular momentum $J_{\text{max}} = 30$.
The uniformity of the error in the $q > m_\pi$ region motivated using the root-mean-square error (RMSE), which is sensitive to outliers like those seen below m_π. I find an RMSE of 0.5 – 2.0 mb for $n \leq 64$ wave packets. The matrix inversion method achieves a similar RMSE for basis sizes $n \approx 16$ and with $n_E \in [16, 32]$.
One should also note that the WPCD method RMSE decreases very slowly for $n > 64$, whereas the matrix inversion RMSE decreases exponentially with n. For reference, a Bayesian analysis of chiral interactions suggests that the χEFT truncation-error in the $q > m_\pi$ region is, at best, 2 mb at around a 68% degree-of-belief (DoB) and 5 mb at 95% DoB [25].

The GPU acceleration of the WPCD method was implemented with the CUDA interface for the use of Nvidia GPUs. The efficiency of parallel algorithms can be held back by extensive read/write access to global hardware memory. In the case of Nvidia GPUs, this memory traffic is alleviated by use of the on-chip shared memory. In this thesis I employed a matrix-matrix multiplication algorithm to explicitly demonstrate the importance of using shared memory.
The GPU WPCD implementation exhibits a tenfold speedup at each basis size n, compared to the threaded CPU-implementation of the matrix inversion method with $n_E = 8$. The matrix inversion time scales linearly with n_E. Since the two methods exhibit different RMSE at a given combination of n and n_E, I also studied the time profile of the two implementations at comparable RMSEs from calculating total NN cross sections. In the $q > m_\pi$ region, the CPU-implementation of the matrix inversion method requires 8 – 20 ms for RMSEs below 3.0 mb, again with a partial-wave

expansion truncation at $J_{\max} = 30$, whereas the WPCD implementation used 0.5 – 12 ms. I emphasise here that the 0.5 ms GPU WPCD profile was done for $n = 16$, yielding 2.0 mb RMSE, for which the matrix inversion method used ~ 15 ms. Therefore, if a 2.0 mb RMSE is deemed as an acceptable method error in a Bayesian parameter estimation then the GPU accelerated WPCD method can calculate scattering observables 30 times faster compared to the matrix inversion method. Note that this is the best-case scenario seen in this study.
In terms of the respective arithmetic complexity models, the efficiency of the GPU implementation of the WPCD method is comparable to the CPU implementation of the matrix inversion method.

The WPCD method error is a limiting factor when striving to speed up scattering observable calculations. However, there remain very interesting open questions in regard to reducing the method error. Specifically, changing the distribution of wave-packet boundaries to improve the approximation of EQ weights. One idea is to combine EQ with Gaussian quadrature by setting the wave-packet bin widths according to a Gauss-Legendre distribution. Other interesting distributions to investigate would be for example Hermite or Laguerre polynomials.

There exists extensive ongoing research on GPU technology. Small hardware changes can drastically increase GPU acceleration. For example, in the transition from CUDA 9 to CUDA 10 the cuSOLVER [73] library routine for the Jacobi method was modified to allow for shared memory usage for square matrices with size greater than 32×32, thus greatly improving the parallel WPCD method performance. Similarly, hardware or software modifications to allow for more efficient memory usage, e.g. increased on-chip memory cache or increased memory broadband to the global GPU memory, should improve the profiles shown in this thesis.

Having the WPCD calculation of NN scattering observables, it is highly interesting to look into three-nucleon (3N) scattering observables next. It is known that 3N forces are necessary for a realistic prediction of the properties of atomic nuclei and their reactions, see e.g. [76–78]. For χEFT approaches, it is therefore highly relevant to study the chiral 3N interaction and associated truncation uncertainties. The advantage of analysing 3N scattering observables is that they provide a direct handle on the momentum dependence of 3N forces. This strongly motivates an interest in developing a method for efficient 3N elastic scattering calculations.
Numerically solving the necessary Faddeev equations [79] is a challenging task [33, 80–84]. However, 3N elastic scattering is where the most promising aspects of the

WPCD method lie [28, 31, 54]. The simplifications already seen for WPCD method in the LS equation reappear in the Faddeev equation [85]. The complicated pole terms of the free 3N propagator are averaged out similarly to the resolvent pole averaging used in the LS equation here. The computational inefficiency of diagonalising small (e.g. $10^2 \times 10^2$) Hamiltonian matrices on the GPU is expected to improve with the larger matrices (e.g. $10^3 \times 10^3$) that appear in the Faddeev equations. GPU acceleration should be quite profitable for solving the Faddeev equations [86, 87], and is therefore a very promising avenue for future research.

Appendix A

Calculating the Resolvent in a Wave-Packet Basis

The nomenclature used here is taken from chapter 3. The resolvent $g(E)$ for the full system Hamiltonian $\hat{h} \equiv \hat{h}_0 + \hat{v}$,

$$\hat{g}(E) \equiv \frac{1}{E - \hat{h} \pm i\epsilon} \,, \tag{A.1}$$

can be calculated analytically in a pseudostate wave-packet basis $\{|z_i\rangle\}_{i=1}^{n}$ of the system Hamiltonian. The resolvent is represented in the basis by (see equation 3.14)

$$\begin{aligned} \langle z_i|\hat{g}(E)|z_j\rangle &= \langle z_i|\frac{1}{E - \hat{h} \pm i\epsilon}|z_j\rangle \\ &= \frac{1}{\mu\sqrt{D_i D_j}} \int_{\mathcal{D}_i}\int_{\mathcal{D}_j} \mathrm{d}p\,\mathrm{d}p' \, \frac{pp'\sqrt{pp'}\langle p|p'\rangle}{E - \frac{p'^2}{2\mu} \pm i\epsilon} \,, \end{aligned} \tag{A.2}$$

where $|p\rangle$ and $|p'\rangle$ are a radial momentum-states. Note that we set the weight function $f(p) = \sqrt{\frac{p}{\mu}}$ and normalisation $N_i = \sqrt{D_i}$ as these are energy wave-packets (from the diagonalisation of $\hat{h}$). From equation 3.15 we know this can be written as

$$\langle z_i|\hat{g}(E)|z_j\rangle = \frac{\delta_{ij}}{\mu D_i} \int_{\mathcal{D}_i} \mathrm{d}p \, \frac{p}{E - \frac{p^2}{2\mu} \pm i\epsilon} \,, \tag{A.3}$$

where we have introduced the Kronecker delta δ_{ij} since $\langle p|p'\rangle = 0 \,\forall\, \mathcal{D}_i \neq \mathcal{D}_j$. Using $E = \frac{q^2}{2\mu}$, where q is the on-shell c.m.s. momentum, we get

$$\langle z_i|\hat{g}(E)|z_j\rangle = \frac{2\delta_{ij}}{D_i} \int_{\mathcal{D}_i} \mathrm{d}p \, \frac{p}{q^2 - p^2 \pm i\epsilon} \,, \tag{A.4}$$

If $E \notin \mathcal{D}_i$, we take the limit $\epsilon \to 0$ and solve the integral to find

$$\langle z_i|\hat{g}(E)|z_j\rangle = \frac{2\delta_{ij}}{D_i}\int_{\mathcal{D}_i} \mathrm{d}p\, \frac{p}{q^2-p^2} \tag{A.5}$$

$$= \frac{2\delta_{ij}}{D_i}\frac{1}{2q}\left[\ln\left|\frac{p}{q}+1\right| - \ln\left|1-\frac{p}{q}\right|\right]_{q_i}^{q_{i+1}} \tag{A.6}$$

$$= \frac{\delta_{ij}}{qD_i}\left[\ln\left|\frac{q+q_i}{q+q_{i-1}}\right| + \ln\left|\frac{q-q_{i-1}}{q-q_i}\right|\right] \quad ; \quad q \notin \mathcal{D}_i\,, \tag{A.7}$$

where q_i and q_{i+1} is the lower and upper boundary of $\mathcal{D}_i$ expressed in momentum, respectively. If $E \in \mathcal{D}_i$, then we have a simple pole at $q = p$. The pole-integration is done using the infinitesimal complex rotation $\pm i\epsilon$ together with the residue theorem, giving

$$\langle z_i|\hat{g}(E)|z_j\rangle = \frac{\delta_{ij}}{qD_i}\left[\ln\left|\frac{q+q_{i+1}}{q+q_i}\right| + \ln\left|\frac{q-q_i}{q-q_{i+1}}\right| - i\pi(\theta(q-q_i)) - \theta(q-q_{i+1})\right]\,, \tag{A.8}$$

where θ is the Heaviside step-function. This is similar the free resolvent for moment wave-packets shown in [28], but the reason we get the same for energy wave-packets is due to our states $|q\rangle$ being radial momentum-states.
The derivation of the free resolvent expressed in a free wave-packet representation follows a similar derivation. In that scenario, it is possible to use momentum wave-packets where $f(p) = 1$ and $N_i = \sqrt{d_i}$, in which case the derivation above changes a little.

Energy averaging of the resolvent is done by integrating the resolvent with respect to E, in the bin $E \in \mathcal{D}_k$, divided by the bin width D_k. We introduce the denotation $\bar{g}^k_{ij}$ to reflect this. The derivation is straightforward (note $g^k_i \equiv \bar{g}^k_{ii}$ in equation 3.34),

$$\begin{aligned}\bar{g}^k_{ij} &\equiv \frac{1}{D_k}\int_{\mathcal{D}_k} \mathrm{d}E\, \langle z_i|\hat{g}(E)|z_j\rangle \\ &= \frac{\delta_{ij}}{\mu D_k D_i}\int_{\mathcal{D}_k} \mathrm{d}q\, \left[\ln\left|\frac{q+q_{i+1}}{q+q_i}\right| + \ln\left|\frac{q-q_i}{q-q_{i+1}}\right| - i\pi\delta_{ik}\right] \\ &= \frac{\delta_{ij}}{D_i N_i}\left[W^+_{ki} - W^-_{ki}\right] - \frac{i\pi}{D_k}\delta_{ik}\,,\end{aligned} \tag{A.9}$$

where,

$$W^{\pm}_{ki} \equiv \sum_{k'=k}^{k+1}\sum_{i'=i}^{i+1}(-1)^{k-k'+i-i'}\left[q_{k'} \pm q_{i'}\right]\ln\left|q_{k'} \pm q_{i'}\right|\,. \tag{A.10}$$

Bibliography

1. Carlson, J. *et al.* Quantum Monte Carlo methods for nuclear physics. *Rev. Mod. Phys.* **87,** 1067–1118. https://link.aps.org/doi/10.1103/RevModPhys.87.1067 (3 Sept. 2015).
2. Hagen, G., Papenbrock, T, Hjorth-Jensen, M. & Dean, D. J. Coupled-cluster computations of atomic nuclei. *Rep. Prog. Phys.* **77,** 096302. https://doi.org/10.1088%2F0034-4885%2F77%2F9%2F096302 (Sept. 2014).
3. Barrett, B. R., Navratil, P. & Vary, J. P. Ab initio no core shell model. *Prog. Part. Nucl. Phys.* **69,** 131–181 (2013).
4. Tews, I., Davoudi, Z., Ekström, A., Holt, J. D. & Lynn, J. E. *New Ideas in Constraining Nuclear Forces* in (Jan. 2020). arXiv: 2001.03334 [nucl-th].
5. Yukawa, H. On the Interaction of Elementary Particles I. *Proc. Phys. Math. Soc. Jap.* **17,** 48–57 (1935).
6. Machleidt, R. in *Advances in Nuclear Physics* (eds Negele, J. W. & Vogt, E.) 189–376 (Springer US, Boston, MA, 1989). ISBN: 978-1-4613-9907-0. https://doi.org/10.1007/978-1-4613-9907-0_2.
7. Weinberg, S. Phenomenological Lagrangians. *Physica A: Statistical Mechanics and its Applications* **96,** 327 –340. ISSN: 0378-4371. http://www.sciencedirect.com/science/article/pii/0378437179902231 (1979).
8. Fritzsch, H., Gell-Mann, M. & Leutwyler, H. Advantages of the color octet gluon picture. *Phys. Lett. B* **47,** 365 –368. ISSN: 0370-2693. http://www.sciencedirect.com/science/article/pii/0370269373906254 (1973).
9. Yang, C. N. & Mills, R. L. Conservation of Isotopic Spin and Isotopic Gauge Invariance. *Phys. Rev.* **96,** 191–195. https://link.aps.org/doi/10.1103/PhysRev.96.191 (1 Oct. 1954).

10. Coleman, S., Wess, J. & Zumino, B. Structure of Phenomenological Lagrangians. I. *Phys. Rev.* **177,** 2239–2247. https://link.aps.org/doi/10.1103/PhysRev.177.2239 (5 Jan. 1969).

11. Callan, C. G., Coleman, S., Wess, J. & Zumino, B. Structure of Phenomenological Lagrangians. II. *Phys. Rev.* **177,** 2247–2250. https://link.aps.org/doi/10.1103/PhysRev.177.2247 (5 Jan. 1969).

12. Gasser, J & Leutwyler, H. Chiral perturbation theory to one loop. *Ann. Phys. (N. Y.)* **158,** 142 –210. ISSN: 0003-4916. http://www.sciencedirect.com/science/article/pii/0003491684902422 (1984).

13. Gasser, J. & Leutwyler, H. Chiral perturbation theory: Expansions in the mass of the strange quark. *Nucl. Phys. B* **250,** 465 –516. ISSN: 0550-3213. http://www.sciencedirect.com/science/article/pii/0550321385904924 (1985).

14. Gasser, J., Sainio, M. & Švarc, A. Nucleons with chiral loops. *Nucl. Phys. B* **307,** 779 –853. ISSN: 0550-3213. http://www.sciencedirect.com/science/article/pii/0550321388901083 (1988).

15. Weinberg, S. Nuclear forces from chiral lagrangians. *Phys. Lett. B* **251,** 288 –292. ISSN: 0370-2693. http://www.sciencedirect.com/science/article/pii/0370269390909383 (1990).

16. Weinberg, S. Effective chiral lagrangians for nucleon-pion interactions and nuclear forces. *Nucl. Phys. B* **363,** 3 –18. ISSN: 0550-3213. http://www.sciencedirect.com/science/article/pii/055032139190231L (1991).

17. Weinberg, S. Three-body interactions among nucleons and pions. *Phys. Lett. B* **295,** 114 –121. ISSN: 0370-2693. http://www.sciencedirect.com/science/article/pii/037026939290099P (1992).

18. Epelbaum, E. & Meißner, U.-G. On the Renormalization of the One–Pion Exchange Potential and the Consistency of Weinberg's Power Counting. *Few-Body Systems* **54,** 2175–2190. ISSN: 1432-5411. https://doi.org/10.1007/s00601-012-0492-1 (2013).

19. Phillips, D. R. Recent results in chiral effective field theory for the NN system. *PoS* **CD12** (eds Goity, J. & Chen, J.) 013. arXiv: 1302.5959 [nucl-th] (2013).

20. Grießhammer, H. W. Assessing Theory Uncertainties in EFT Power Countings from Residual Cutoff Dependence. *PoS* **CD15,** 104. arXiv: 1511.00490 [nucl-th] (2016).

21. Nogga, A., Timmermans, R. G. E. & Kolck, U. v. Renormalization of one-pion exchange and power counting. *Phys. Rev. C* **72,** 054006. https://link.aps.org/doi/10.1103/PhysRevC.72.054006 (5 2005).

22. Valderrama, M. P. Power counting and Wilsonian renormalization in nuclear effective field theory. *Int. J. Mod. Phys. E* **25,** 1641007. eprint: https://doi.org/10.1142/S021830131641007X. https://doi.org/10.1142/S021830131641007X (2016).

23. Brooks, S., Gelman, A., Jones, G. & Meng, X. *Handbook of Markov Chain Monte Carlo* ISBN: 9781420079425 (CRC Press, 2011).

24. Oakley, A. & O'Hagan, A. Bayesian inference for the uncertainty distribution of computer model outputs. *Biometrika* **89,** 769–784. ISSN: 0006-3444. eprint: https://academic.oup.com/biomet/article-pdf/89/4/769/667202/890769.pdf. https://doi.org/10.1093/biomet/89.4.769 (Dec. 2002).

25. Melendez, J. A., Wesolowski, S. & Furnstahl, R. J. Bayesian truncation errors in chiral effective field theory: Nucleon-nucleon observables. *Phys. Rev. C* **96,** 024003. https://link.aps.org/doi/10.1103/PhysRevC.96.024003 (2 Aug. 2017).

26. Carlsson, B. D. *et al.* Uncertainty Analysis and Order-by-Order Optimization of Chiral Nuclear Interactions. *Phys. Rev. X* **6,** 011019. https://link.aps.org/doi/10.1103/PhysRevX.6.011019 (1 Feb. 2016).

27. Furnstahl, R. J., Klco, N., Phillips, D. R. & Wesolowski, S. Quantifying truncation errors in effective field theory. *Phys. Rev. C* **92,** 024005. https://link.aps.org/doi/10.1103/PhysRevC.92.024005 (2 Aug. 2015).

28. Rubtsova, O., Kukulin, V. & Pomerantsev, V. Wave-packet continuum discretization for quantum scattering. *Ann. Phys. (N. Y.)* **360,** 613 –654. ISSN: 0003-4916. http://www.sciencedirect.com/science/article/pii/S0003491615001773 (2015).

29. Haftel, M. I. & Tabakin, F. Nuclear saturation and the smoothness of nucleon-nucleon potentials. *Nucl. Phys. A* **158,** 1 –42. ISSN: 0375-9474. http://www.sciencedirect.com/science/article/pii/0375947470900473 (1970).

30. Müther, H., Rubtsova, O. A., Kukulin, V. I. & Pomerantsev, V. N. Discrete wave-packet representation in nuclear matter calculations. *Phys. Rev. C* **94,** 024328. https://link.aps.org/doi/10.1103/PhysRevC.94.024328 (2 Aug. 2016).

31. Pomerantsev, V. N., Kukulin, V. I. & Rubtsova, O. A. Solving three-body scattering problems in the momentum lattice representation. *Phys. Rev. C* **79,** 034001. https://link.aps.org/doi/10.1103/PhysRevC.79.034001 (3 Mar. 2009).

32. Lippmann, B. A. & Schwinger, J. Variational Principles for Scattering Processes. I. *Phys. Rev.* **79,** 469–480. https://link.aps.org/doi/10.1103/PhysRev.79.469 (3 Aug. 1950).

33. Glöckle, W. *The quantum mechanical few-body problem* (Springer Science & Business Media, 2012).

34. Bystricky, J., Lehar, F. & Winternitz, P. Formalism of nucleon-nucleon elastic scattering experiments. *J. Phys. France* **39,** 1–32. https://doi.org/10.1051/jphys:019780039010100 (1978).

35. Hoshizaki, N. Formalism of nucleon-nucleon scattering. *Prog. Theor. Phys. Suppl.* **42,** 107–159 (1968).

36. Wolfenstein, L. & Ashkin, J. Invariance Conditions on the Scattering Amplitudes for Spin ½ Particles. *Phys. Rev.* **85,** 947–949. https://link.aps.org/doi/10.1103/PhysRev.85.947 (6 Mar. 1952).

37. Wolfenstein, L. Possible Triple-Scattering Experiments. *Phys. Rev.* **96,** 1654–1658. https://link.aps.org/doi/10.1103/PhysRev.96.1654 (6 Dec. 1954).

38. Wolfenstein, L. Charge Independence and Polarization Experiments. *Phys. Rev.* **101,** 427–428. https://link.aps.org/doi/10.1103/PhysRev.101.427 (1 Jan. 1956).

39. Wolfenstein, L. Polarization of fast nucleons. *Ann. Rev. Nucl. Part. Sci.* **6,** 43–76 (1956).

40. Sakurai, J. J. & Napolitano, J. *Modern Quantum Mechanics* 2nd ed. (Cambridge University Press, 2017).

41. Erkelenz, K., Alzetta, R. & Holinde, K. Momentum space calculations and helicity formalism in nuclear physics. *Nucl. Phys. A* **176,** 413–432 (1971).

42. Blatt, J. M. & Biedenharn, L. C. The Angular Distribution of Scattering and Reaction Cross Sections. *Rev. Mod. Phys.* **24,** 258–272. https://link.aps.org/doi/10.1103/RevModPhys.24.258 (4 Oct. 1952).

43. Stapp, H. P., Ypsilantis, T. J. & Metropolis, N. Phase-Shift Analysis of 310-Mev Proton-Proton Scattering Experiments. *Phys. Rev.* **105,** 302–310. https://link.aps.org/doi/10.1103/PhysRev.105.302 (1 Jan. 1957).

44. Croz, J. J. d. & Higham, N. J. Stability of Methods for Matrix Inversion. *IMA J. Numer. Anal.* **12,** 1–19. ISSN: 0272-4979. eprint: https://academic.oup.com/imajna/article-pdf/12/1/1/6767713/12-1-1.pdf. https://doi.org/10.1093/imanum/12.1.1 (Jan. 1992).

45. Heller, E. J. Theory of J-matrix Green's functions with applications to atomic polarizability and phase-shift error bounds. *Phys. Rev. A* **12,** 1222–1231. https://link.aps.org/doi/10.1103/PhysRevA.12.1222 (4 1975).

46. Winick, J. R. & Reinhardt, W. P. Moment T-matrix approach to e^+-H scattering. I. Angular distribution and total cross section for energies below the pickup threshold. *Phys. Rev. A* **18,** 910–924. https://link.aps.org/doi/10.1103/PhysRevA.18.910 (3 1978).

47. Winick, J. R. & Reinhardt, W. P. Moment T-matrix approach to e^+-H scattering. II. Elastic scattering and total cross section at intermediate energies. *Phys. Rev. A* **18,** 925–934. https://link.aps.org/doi/10.1103/PhysRevA.18.925 (3 1978).

48. Heller, E. J., Rescigno, T. N. & Reinhardt, W. P. Extraction of Scattering Information from Fredholm Determinants Calculated in an L^2 Basis: A Chebyschev Discretization of the Continuum. *Phys. Rev. A* **8,** 2946–2951. https://link.aps.org/doi/10.1103/PhysRevA.8.2946 (6 1973).

49. Corcoran, C. T. & Langhoff, P. W. Moment-theory approximations for nonnegative spectral densities. *J. Math. Phys.* **18,** 651–657. https://doi.org/10.1063/1.523321 (1977).

50. Weyl, H. Über gewöhnliche Differentialgleichungen mit Singularitäten und die zugehörigen Entwicklungen willkürlicher Funktionen. *Math. Ann.* **68,** 220–269 (June 1910).

51. Bethe, H. in *Quantentheorie* (eds Bethe, H. *et al.*) 273–560 (Springer Berlin Heidelberg, Berlin, Heidelberg, 1933). ISBN: 978-3-642-52619-0. https://doi.org/10.1007/978-3-642-52619-0_3.

52. Wigner, E. P. *Group theory and its application to the quantum mechanics of atomic spectra* Trans. from the German. https://cds.cern.ch/record/102713 (Academic Press, New York, NY, 1959).

53. Rubtsova, O. A. & Kukulin, V. I. Wave-packet discretization of a continuum: Path toward practically solving few-body scattering problems. *Phys. At. Nucl.* **70,** 2025–2045. ISSN: 1562-692X. https://doi.org/10.1134/S1063778807120058 (2007).

54. Kukulin, V. I., Pomerantsev, V. N. & Rubtsova, O. A. Wave-packet continuum discretization method for solving the three-body scattering problem. *Theor. Math. Phys.* **150,** 403–424. ISSN: 1573-9333. https://doi.org/10.1007/s11232-007-0030-3 (2007).

55. Rubtsova, O. A., Pomerantsev, V. N. & Kukulin, V. I. Quantum scattering theory on the momentum lattice. *Phys. Rev. C* **79,** 064602. https://link.aps.org/doi/10.1103/PhysRevC.79.064602 (6 June 2009).

56. Kukulin, V. I. & Rubtsova, O. A. Discrete Quantum Scattering Theory. *Theor. Math. Phys.* **134,** 404–426. ISSN: 1573-9333. https://doi.org/10.1023/A:1022657607306 (2003).

57. Golub, G. H. & Van Loan, C. F. *Matrix Computations* Third (The Johns Hopkins University Press, Baltimore, MD, USA, 1996).

58. Bae, S. E., Shinn, T.-W. & Takaoka, T. A Faster Parallel Algorithm for Matrix Multiplication on a Mesh Array. *Procedia Computer Science* **29.** 2014 International Conference on Computational Science, 2230 –2240. ISSN: 1877-0509. http://www.sciencedirect.com/science/article/pii/S1877050914003858 (2014).

59. C3SE. *Chalmers Centre for Computational Science and Engineering* https://www.c3se.chalmers.se/ (2020).

60. Jia, Z., Maggioni, M., Smith, J. & Scarpazza, D. P. *Dissecting the NVidia Turing T4 GPU via Microbenchmarking* 2019. arXiv: 1903.07486 [cs.DC].

61. Nvidia. *NVIDIA K80/K40* https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/TeslaK80-datasheet.pdf (2020).

62. Nvidia. *NVIDIA T4* https://www.nvidia.com/en-us/data-center/tesla-t4/ (2020).

63. Nvidia. *NVIDIA V100 TENSOR CORE GPU* https://www.nvidia.com/en-us/data-center/v100/ (2020).

64. Intel. *Intel® Xeon® Processor E5-2650 v3* https://ark.intel.com/content/www/us/en/ark/products/81705/intel-xeon-processor-e5-2650-v3-25m-cache-2-30-ghz.html (2020).

65. *Intel Xeon E5 v3 Product Family; Specification Update* 330785-011. Intel (Sept. 2017).

66. Intel. *Intel® Xeon® Gold 6130 Processor* https://ark.intel.com/content/www/us/en/ark/products/120492/intel-xeon-gold-6130-processor-22m-cache-2-10-ghz.html (2020).

67. Wilt, N. *Cuda Handbook: A Comprehensive Guide to Gpu Programming* ISBN: 9781548588045 (CreateSpace Independent Publishing Platform, 2017).

68. Matthes, A. *Example: Matrix multiplication* http://texample.net/tikz/examples/matrix-multiplication/ (2020).

69. Ekström, A. *et al.* Optimized Chiral Nucleon-Nucleon Interaction at Next-to-Next-to-Leading Order. *Phys. Rev. Lett.* **110,** 192502. https://link.aps.org/doi/10.1103/PhysRevLett.110.192502 (19 May 2013).

70. Levinson, N. On the uniqueness of the potential in a Schrodinger equation for a given asymptotic phase. *Kgl. Danske Videnskab Selskab. Mat. Fys. Medd.*

71. Epelbaum, E., Hammer, H.-W. & Meißner, U.-G. Modern theory of nuclear forces. *Rev. Mod. Phys.* **81,** 1773–1825. https://link.aps.org/doi/10.1103/RevModPhys.81.1773 (4 Dec. 2009).

72. Nvidia. *Dense Linear Algebra on GPUs* https://developer.nvidia.com/cublas (2020).

73. Nvidia. *CUDA Toolkit Documentation* https://docs.nvidia.com/cuda/cusolver/index.html (2020).

74. Machleidt, R. & Entem, D. *Chiral Symmetry and the Nucleon-Nucleon Interaction* in (2011), 317–343. arXiv: 1110.3022 [nucl-th].

75. Machleidt, R. & Entem, D. Chiral effective field theory and nuclear forces. *Phys. Rep.* **503,** 1 –75. ISSN: 0370-1573. http://www.sciencedirect.com/science/article/pii/S0370157311000457 (2011).

76. Otsuka, T., Suzuki, T., Holt, J. D., Schwenk, A. & Akaishi, Y. Three-Body Forces and the Limit of Oxygen Isotopes. *Phys. Rev. Lett.* **105,** 032501. https://link.aps.org/doi/10.1103/PhysRevLett.105.032501 (3 July 2010).

77. Hagen, G., Hjorth-Jensen, M., Jansen, G. R., Machleidt, R. & Papenbrock, T. Continuum Effects and Three-Nucleon Forces in Neutron-Rich Oxygen Isotopes. *Phys. Rev. Lett.* **108,** 242501. https://link.aps.org/doi/10.1103/PhysRevLett.108.242501 (24 June 2012).

78. Hammer, H.-W., Nogga, A. & Schwenk, A. Colloquium: Three-body forces: From cold atoms to nuclei. *Rev. Mod. Phys.* **85,** 197–217. https://link.aps.org/doi/10.1103/RevModPhys.85.197 (1 Jan. 2013).

79. Faddeev, L. Scattering theory for a three particle system. *Zh. Eksp. Teor. Fiz.* **39,** 1014–1019 (1960).

80. Deltuva, A., Fonseca, A. C. & Sauer, P. U. Momentum-space description of three-nucleon breakup reactions including the Coulomb interaction. *Phys. Rev. C* **72,** 054004. https://link.aps.org/doi/10.1103/PhysRevC.72.054004 (5 Nov. 2005).

81. Epelbaum, E. *et al.* Three-nucleon forces from chiral effective field theory. *Phys. Rev. C* **66,** 064001. https://link.aps.org/doi/10.1103/PhysRevC.66.064001 (6 Dec. 2002).

82. Witała, H. *et al.* Nd elastic scattering as a tool to probe properties of $3N$ forces. *Phys. Rev. C* **63,** 024007. https://link.aps.org/doi/10.1103/PhysRevC.63.024007 (2 Jan. 2001).

83. Glöckle, W. in *Scattering* (eds Pike, R. & Sabatier, P.) 1339 –1359 (Academic Press, London, 2002). ISBN: 978-0-12-613760-6. http://www.sciencedirect.com/science/article/pii/B9780126137606500723.

84. Glöckle, W., Witała, H., Hüber, D., Kamada, H. & Golak, J. The three-nucleon continuum: achievements, challenges and applications. *Phys. Rep.* **274,** 107 – 285. ISSN: 0370-1573. http://www.sciencedirect.com/science/article/pii/0370157395000852 (1996).

85. Kukulin, V. I. & Rubtsova, O. A. New Way in Few-Body Scattering Calculations. *Few-Body Systems* **54,** 1611–1615. ISSN: 1432-5411. https://doi.org/10.1007/s00601-013-0628-y (2013).

86. Pomerantsev, V., Kukulin, V. & Rubtsova, O. New general approach in few-body scattering calculations: Solving discretized Faddeev equations on a graphics processing unit. *Phys. Rev. C* **89,** 064008. arXiv: 1404.5253 [nucl-th] (2014).

87. Rubtsova, O., Pomerantsev, V. & Kukulin, V. *Solving Few-Body Scattering Problems in a Discrete Representation by Using GPU* in *5th International Conference Nuclear Theory in the Supercomputing Era* (Apr. 2018), 205–225.